Building Interacting Tutorials for Teaching Psychological Statistics Online with *learnr*

Chris Aberson
Humboldt State University

ABSTRACT

Faculty are increasingly tasked with online teaching duties. This paper provides a how-to guide to using the *learnr* package for R. This package allows instructors to create seamless interactive tutorials that can use video, quizzes, and exercises run in R to foster student engagement and learning of statistics. Since the beginning of the Covid-19 pandemic, more faculty than ever are teaching online. *learnr* tutorials provide a format that allows for greater student engagement with materials by providing opportunities to test knowledge and practice after viewing short videos on topics. I provide concrete instructions for developing learnr tutorials for teaching introductory statistics and provide insights from having applied these technologies for over a year. learnr is a tool that can assist instructors in leveraging valuable teaching opportunities afforded by the technology is a manner that requires only small changes in their more prevalent approaches to teaching online.

Keywords: Statistics, R, Interactive Tutorial, *learnr*

1. INTRODUCTION

The *learnr* package was developed by members of the RStudio team (RStudio Team 2020) to provide an easy-to-use approach to building interactive tutorials. In this tutorial, I demonstrate strategies for using *learnr* to build interactive tutorials and provide examples of how to develop materials using this tool. Please see (https://github.com/chrisaberson/IntroStatsTutorials) for the most recent version of these tutorials.

Most faculty experienced a considerable increase in online teaching responsibilities due to the Covid-19 pandemic. Although such changes are challenging, these new responsibilities bring considerable opportunity to re-evaluate course delivery. The *learnr* package for R provides considerable tools for using existing technology to move away from a simple lecture focus to a more dynamic approach that mixes short instructional videos with low stakes quizzes and a space to complete exercising using R.

The intended audience for this paper are those teaching statistics in any discipline with R and RStudio used jointly as the primary statistical software. The *learnr* package only works within the RStudio environment. I built all materials using R version 4.03 and RStudio version 1.4.904. Readers not familiar with R, RStudio, or R Markdown, will find useful online resources at https://r4ds.had.co.nz/ (a free text on using R) and https://www.rstudio.com/resources/webinars/ (RStudio free webinars).

In my own Introductory Psychological Statistics course, I design each tutorial around a topic (e.g., data visualization, t-tests) and provide materials that include detailed learning objectives, numerous short (usually 5-10 minute) videos on topics, interactive (ungraded) quizzes, apps demonstrating concepts, and data-based exercises. The student's experience in working through tutorials is one of watching a short video, taking a quiz on what they just watched, seeing detailed examples of code, working through exercises using R, and then completing a quiz focused on interpretation of the statistical results. In contrast to a typical course wherein students sit through lecture and then move to a laboratory session to get familiar with software-driven analyses, the tutorial approach makes working with materials far more immediate while providing a low-stakes environment for learning. *learnr* tutorials were the primary means of content delivery. Assessments primarily involved homework assignments, exams, and a final project.

This approach provides several advantages. First, use of low-stakes assessment enhances retrieval of information. In addition, use of these assessments relates to improved performances on knowledge assessments taken later (Sana et al. 2021). Tutorials are a form of self-regulated study. Self-regulated study has many benefits including accurate assessments of ongoing learning and a better understanding of how learning happens (Kornell and Bjork 2007).

In this paper, I detail how to get started with developing *learnr* tutorials, how to bundle tutorials and data into an R package for distribution, and provide tips based on my experiences having nearly 100 beta testers (a.k.a., students) use my tutorials across various statistics courses. This article is meant to supplement tutorial materials found at https://rstudio.github.io/learnr/ by making them more accessible and providing detailed "how to" examples.

2. MAKING THE TUTORIAL

As a first step, install the *learnr* package (Schloerke et al. 2020). Choose file – new file – R Markdown – From Template to create a new tutorial. Alternatively, it can be simpler to take an existing tutorial and modify it. Some useful examples can be found at https://github.com/chrisaberson/IntroStatsTutorials and https://github.com/profandyfield/adventr.

Next, begin to add elements. To do so, requires setting various parameters. In R Markdown, code is placed in "chunks." Chunks are distinguished from text as they include executable code. Choosing "From Template" when creating a new file will create a file with an automatically generated header as in Code Snippet 1. In general, it is best not to change this header as it is fussy. Every space or dash matters.

Code Snippet 1 demonstrates the automatically generated header.

```
---
title: "Data Visualization"
output: learnr::tutorial
runtime: shiny_prerendered
description: >
  Data Visualization
---
```

Code Snippet 1. *learnr* header

The materials on lines 10,11, and 16 are automatically generated global settings necessary to correctly display quizzes (I come back to these later). Add the necessary libraries to your tutorial by simply typing code inside the chunk. Code Snippet 2 adds the libraries *learnr*, *ggplot2*, *knitr* and *IntroStatsTutorials*

```
10 ```{r setup, include=FALSE}
11 knitr::opts_chunk$set(echo = FALSE)
12 library(learnr)
13 library(ggplot2)
14 library(knitr)
15 library(IntroStatsTutorials)
16 ```
```

Code Snippet 2. Opening Code Chunk

Next, add content. You can control the "flow" of the tutorial using ## and ### to define heading levels. The ## heading level means that everything that follows is on a new page. The ### heading level means the materials that follow get their own heading but appear on the same page. The # heading level is the title of the tutorial (you only use it once). Code Snippet 3 shows use of headings and the addition of text. As part of my overview, I provide some background on data used for exercises, required packages, and learning objectives.

```
18 # Data Visualization
19
20 ## Overview
21
22 This tutorial focuses on data
   visualization (i.e., graphing) The
   tutorial includes a combination of
```

Code Snippet 3. Top and 2nd level header and text

Figure 1 shows how the tutorial renders the code.

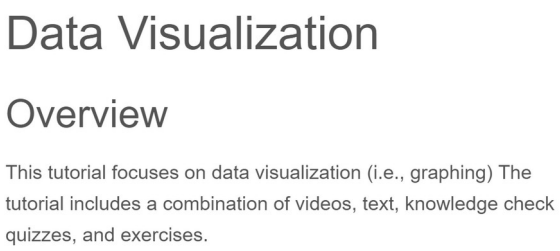

Figure 1. Rendered Code

3. VIDEOS

The code in Code Snippet 4 demonstrates how to add a video. I recommend using short videos that range from 5-9 minutes as this length appears to be most effective at holding student attention (Guo et al. 2014).

```
## Video 1 Introduction to Data Visualization

![Video 1: Introduction](https://youtu.be/ A_Sa9Mp40YE)
```

Code Snippet 4. Adding video

Adding images is a process similar to adding videos. Simply provide the location (ideally in the same directory as your tutorial file with a subdirectory called "images."

4. QUIZZES

After viewing a video, students then proceed to a quiz. The implementation of quizzes is a bit fussy in terms of formatting, so you will end up spending some time figuring out why code is not working. Common issues are missing commas and unbalanced parentheses. As with most statistical programs, R rarely produces error messages that are useful.

learnr quizzes are ungraded. You can decide whether to allow students to try to answer questions multiple times. My approach is to allow students to try until they get the correct answer and provide feedback on each question to further build comprehension. A sample quiz is in Code Snippet 5.

```
```{r quiz1}
learnr::quiz(
  learnr::question("The graph titled 'do you intend to vote' shows which of the following?",
    learnr::answer("Yes is the most common answer", correct = TRUE),
    learnr::answer("No is the most common answer", message = "Only about 20% responded no"),
    learnr::answer("Maybe is the most common answer", message = "Maybe is the least common answer"),
    learnr::answer("None of the above", "I assure you, it is one of the above"),
    correct = "Correct! Yes is the most common response with 85%",
    incorrect = "Sorry, that is incorrect. Try again.",
    random_answer_order = TRUE,
    allow_retry = T
  ),
  learnr::question("The graph titled 'On a scale of 1 to 100 ...' shows ... ",
    learnr::answer("Most answers were between 70 and 90", correct = TRUE),
    learnr::answer("Most answers were between 50 and 100",message="That range appears to
                   capture at least 75% of the scores. Most would mean around 50%"),
    learnr::answer("Most people are not certain they will vote",message="The lower
                   scores on the graph are the less likely to vote people. Not a on of people down there."),
    learnr::answer("None of the above",message="One of the three is correct"),
    correct = "Correct. The graph show that most (i.e., more than half at least) of the scores fall between 70 and 90",
    random_answer_order = TRUE,
    incorrect = "Sorry, that is incorrect. Try again.",
    allow_retry = T
  )
)

```
```

Code Snippet 5. Code for adding a quiz

The chunk begins with a title for the quiz: quiz1. All quizzes include a unique title. The rest of the code is relatively straightforward but there are a handful of components that I want to highlight. First, quizzes of this nature that provide low stakes testing are most useful for students when paired with feedback (Warnock 2013). For correct answers, I make sure to remind the student why their answer is correct. The message command demonstrates how to add feedback.

As noted above, commas and parentheses will typically be the cause of broken quizzes, so do attend to those closely. I copy and paste existing (working) quizzes and then simply modify text for new questions. Third, you can test a quiz by clicking the "play" button in the upper right corner of the chunk.

In writing quizzes, you can provide feedback for correct answers and incorrect answers. To add specific feedback for specific incorrect answers, the message command is used as demonstrated. In writing quizzes, I try to match incorrect answers to common student misconceptions and giving feedback on correct answers that reinforce learning. For example, in one of the questions above, my distractor items focus on common misinterpretations of histograms. One common misinterpretation is confusing the majority of scores with "nearly all the scores." For that option, feedback reads, "that range appears to capture 75% of the scores. 'Most' would mean more than 50%."

5. EXERCISES

Exercises allow users to run analysis in R within the tutorial. My approach to exercises is to first provide an example of how to run an analysis and then have students try those analyses out on a different set of variables or data. The *Code Snippet*s that follow demonstrate a few useful approaches. First, display code, not output. This approach is useful when presenting code to students as a demonstration of how to run analyses. The eval=F in the header tells the tutorial to show the code but not the output. If eval=T then the code will be displayed, and the analysis will run. If echo=F, code is not displayed. I often use this approach to provide output from analyses for quizzes. Code Snippet 6 demonstrates the code.

```
```{r echo = T, eval = F}
hist(denial$PSEUDOSCI, xlab="Pseudoscience Beliefs", main = "Histogram of Pseudoscience Beliefs", col="Pink")
```
```

Code Snippet 6. Running R in the tutorial

To start an exercise, begin with two chunks. One for the question and one for the solution. As with quizzes, exercises require unique names. The first chunk of code (line 255) in Code Snippet 7 establishes that this is an exercise and the second provides the correct syntax. Code Snippet 7 produces the assignment text and a box for students to enter their solution, as rendered in Figure

2.

```
### Exercises: Making Histograms and Density Plots

Using the *denial* data, make a density plot of the climate change denial variable (**CCD**).

```{r ex3, exercise = TRUE, exercise.lines = 1}

```

```{r ex3-solution}
plot(density(denial$CCD))
```
```

Code Snippet 7. An exercise

The code in Code Snippet 7 renders as shown in Figure 2. The Solution button brings up the correct code and the Run Code button submits the code to R. Output, including any error messages, appears directly below.

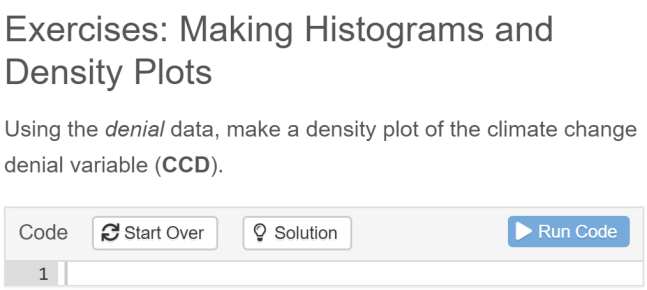

Figure 2. Rendered exercise

One approach that I tend to use with exercises is to combine them with quizzes. A student might complete the exercise portion and then go to a quiz that reproduces their output and focuses on interpretation of the results.

Following my experiences with *learnr*, I offer the following suggestions. If you are using data that is not standard R issue (e.g., iris), you should make a package that includes your data and all other tutorial materials. If you are not using external data, this process is unnecessary. The approach I employ involves creating a package to house the entire course. Test your exercises numerous times. There is nothing more frustrating for students than broken exercises. Clear the R session workspace when testing. I regularly find things only work because of what was active in my workspace, meaning that the tutorial will not run properly in standalone. If, like me, you follow-up exercises with quizzes, be aware that students might not have run their exercise analyses correctly, and so might get the quiz wrong even if they used the correct approach. To account for this, I create a new page for the quiz and run the correct analysis so that the questions address relevant output.

6. TEXT AND EQUATIONS

Text formatting is relatively straightforward with common commands including *italics*, **bold**, Superscript^2^, Subscript~2~. Equations are a bit trickier as they rely on LaTeX. I

found the materials at https://sv.overleaf.com/learn/latex/Mathematical_expressions very helpful for learning LaTeX. Although this format seems challenging at first, I found that an hour or two was all that I needed to learn the format. Code Snippet 8 demonstrates the equation code and how equations render. This code easily accommodates full numeric examples.

```
$\bar{x}$
```
$\bar{x}$

```
$s_{\bar{x_1}-\bar{x_2}} =\sqrt{s_{\bar{x_1}}^2+s_{\bar{x_2}}^2 - 2rs_{\bar{x_1}}s_{\bar{x_2}}}$
```
$s_{\bar{x_1}-\bar{x_2}} =\sqrt{s_{\bar{x_1}}^2+s_{\bar{x_2}}^2 - 2rs_{\bar{x_1}}s_{\bar{x_2}}}$

```
$cov_{xy} = \frac{(x-\bar{x})(y-\bar{y})}{n-1}$
```
$cov_{xy} = \frac{(x-\bar{x})(y-\bar{y})}{n-1}$

```
$r = \frac{cov_{xy}}{s_xs_y}$
```
$r = \frac{cov_{xy}}{s_xs_y}$

```
$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$
```
$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$

Code Snippet 8. Code and rendered equations

7. SHINY APPS

Shiny apps are interactive graphical interfaces that allow for a variety of applications. To the user these are viewed as apps that integrate directly in the tutorial. Pre-made Shiny apps exist for a number of useful visualizations such as guessing correlation size based on a scatterplot and the influence of outliers on correlation. Table 1 provides links to several sources for shiny apps. Inclusion of a Shiny is straightforward, involving only a call to the app as shown in Code Snippet 9. One common issue with Shiny apps is that they only appear when the tutorial is run in a browser (there is a button in the left-hand corner to open in a browser). I have found that I need to experiment with the height parameter within the *include_app* call to make sure everything appears on a single page. (See Code Snippet 9)

Table 1. Shiny app resources

http://facweb.gvsu.edu/adriand1/215apps.html
https://github.com/ShinyEd/intro-stats
https://statistics.calpoly.edu/shiny
https://www4.stat.ncsu.edu/~jbpost2/teaching.html
http://www.artofstat.com/webapps.html
http://facweb.gvsu.edu/adriand1/215apps.html

https://github.com/ShinyEd/intro-stats

```
```{r}
knitr::include_app("https://istats.shinyapps.io/guesscorr/", height = "1000px")
```
```

Code Snippet 9. Adding a shiny app

8. MAKING A PACKAGE FOR DISTRIBUTION

learnr tutorials can be distributed as simple markdown files, however, that approach does not allow for inclusion of datasets. In order to include data, tutorials will need to be packaged. I generally create a single package for the entire class and update it as necessary. I believe that the most challenging aspect of building tutorials is the packaging process. The steps that follow detail how to build a package within RStudio.

The first step in making a package is to create it. Click File – New Project then choose New Directory (best to start empty). Choose R Package using devtools. Give it a name and create the project. At this point you will have a directory with some autogenerated directories. This includes directories for R and man (manuals). To these you will need to add a directory on your computer called "data" and one called "inst" that includes a subdirectory called "tutorials". *Figure 3* demonstrates the directory structures.

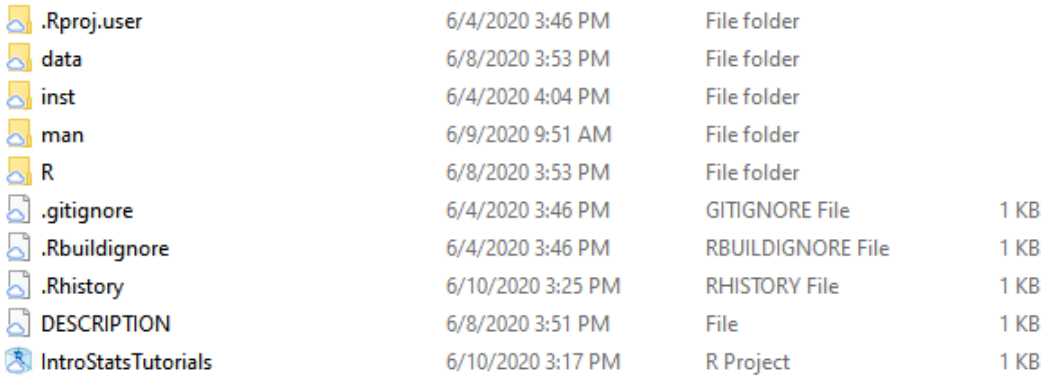

Figure 3. Directory Structure

The next step involves adding a number of specific files. The first is a file inside the R directory with the same name as your package (packagename.R). This file provides some description and, if needed, is where you specify packages used in the tutorial and other pieces. Next is a file inside the R directory called *data.R* that describes your datafiles. If you have large datasets, you might just want to subset these to the variables used in exercises as you have to define all the variables. There will be a file generated automatically called “hello.R” that should be deleted. Finally, go to inst/tutorials/tutorialname and put your markdown file (the tutorial) in that directory. If you have images, put those in inst/tutorials/tutorialname/images.

Code Snippet 10 demonstrates the structure of the data.R file. This file defines all data used in any of the tutorials. Every line is preceded with #'. The @format command defines the dimensions of the datafile and will throw an error if the numbers are wrong. \describe defines each item in order. You have to define everything in the dataset, in the correct order, and spelled correctly. The file name goes on the bottom. If you have multiple datafiles, just repeat the process in the same file.

```
#' Climate Change Denial
#'
#' @format A data frame with 1587 rows and 11 variables:
#' \describe{
#'   \item{CCD}{Climate Change Denial}
#'   \item{ANITESTABL}{Anti-Establishment Beliefs}
#'   \item{EXCL_ANTIEG}{Anti-Egalitarian Preferences}
#'   \item{TRADVALUE}{Traditional Values}
#'   \item{OPENNESS}{Openness}
#'   \item{PSEUDOSCI}{Pseudoscience beliefs}
#'   \item{AGREEABL}{Agreeableness}
#'   \item{Age}{Age in years}
#'   \item{Gender}{Gender}
#'   \item{Education}{Education = Five Categories}
#'   \item{ed}{Education = Two Categories}
#' }
"denial"
```

Code Snippet 10. Structure of the data.R file

Next is the packagename.R file. This provides a brief description of the tutorial. The @import command controls the packages required by the tutorials. Much of the rest of the file is automatically generated. Figure 4 shows the structure of this file.

```
#' Tutorials for Introductory Statistics
#'
#' @description
#'
#' This package supports online teaching for teaching
introductory stats
#'
#' @import ggplot2 car lsr apaTables BayesFactor tidyverse
#' @docType package
#' @name IntroTutorials
#'
#'
#'
NULL
```

Figure 4. Packagename.R file structure

Building your Package

RStudio provides considerable functionality for building packages. Most functionality requires the devtools package (Wickham et al. 2020). There are many approaches to building packages, I present only an RStudio focused approach. Figure 5 shows the basic process for package building. Simply choose Clean and Rebuild to start.

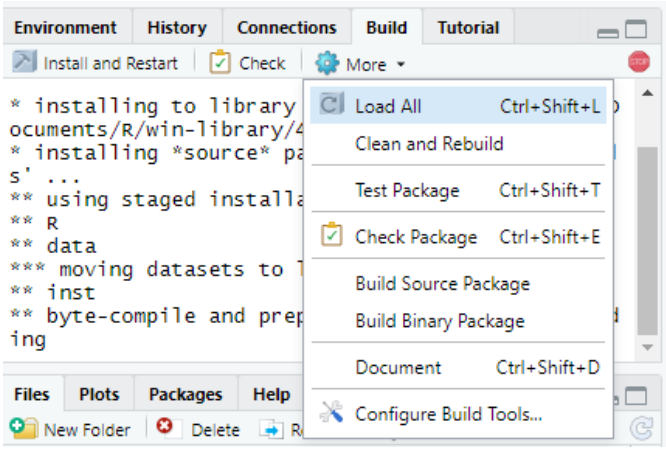

Figure 5. Building a package

You will most likely experience errors when you first build your package. Don't be discouraged, this happens all the time. The error message will provide a line number that corresponds to the chunk that doesn't work; this information will allow you to identify the problematic code chunk.

After building the package, you will have a few new files in your directory. The description file is where you can put basic information about your work. These meta data are generally only necessary for packages that will be widely distributed, so you can skip this step. *Code Snippet* 11 details that basic structure of the description file.

```
Package: IntroStatsTutorials
Title: Tutorials for Intro Statistics
Version: 0.0.0.9000
Authors@R:
    person(given = "Chris",
           family = "Aberson",
           role = c("aut", "cre"),
           email = "cla18@humboldt.edu",
           comment = c(ORCID = "orcid.org/0000-0003-3481-7177"))
Description: Intro Statistics Tutorials.
License: GPL (>= 3)
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.0
Imports:
    car, ggplot2, lsr
```

Code Snippet 11. DESCRIPTION file

Next is the NAMESPACE file. Every package has a file called NAMESPACE. This file controls actions such as the packages that get installed and loaded automatically. Building the package creates an empty NAMESPACE file. *devtools::document()* is a command developed to make building the NAMESPACE simple. Simply type *devtools::document*() with your project open (or use the pull-down menu). The document() command provides a quick way for the NAMESPACE to reflect what was specified in the packagename.R file (when indicating imports). This also generates help files (not useful here but a massive time saver if you are making a package for distribution on CRAN).

Finally, build a source package for distribution. This creates a compressed file that you can easily distribute via LMS or email for students to install. Choosing Build Source Package creates a file called packagename.tar.gz. This file generally gets built to a directory one up from your tutorial. For example, on my computer, this would be G:/My Drive rather than G:/My Drive/IntroStatsTutorials). To do this, students should choose Install from source package and then navigate to the file. Figure 6 includes a screenshot of this process.

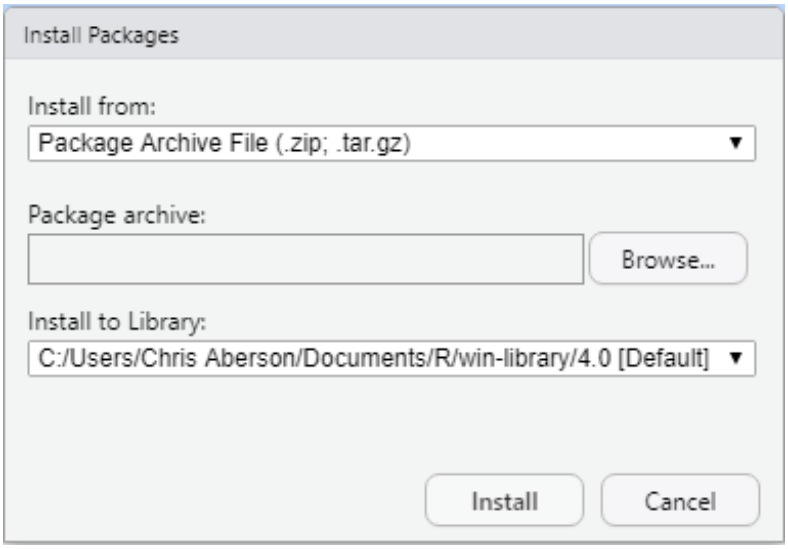

Figure 6. Package installation

9. CHALLENGES AND BEST PRACTICES

As *learnr* and your new packages will be loaded on a large number of individually configured machines across at least three different platforms, there will be occasional errors that pop up. One persistent issue is around packages. Many packages require several other packages. This sometimes creates problems when installing the tutorial. In general, error messages will point to a package that could not be located. A simple fix for this is to download the offending packages manually. Also, as noted before, often times packages will run on your computer but not others. This is usually an issue around having certain required files open in your environment but not having those pieces in the tutorial itself. To address this, I always make sure to clear my environment. Go to Session – Clear Workspace to accomplish this.

To support academic integrity, I add a different randomly generated four-digit number at the end of each student's tutorial. Students need to enter their unique number into our course management software for credit. This approach may not be ideal for all instructors. Those interested in logging student responses for credit should refer to the submitr (https://github.com/dtkaplan/submitr) or learnrhash (https://github.com/rundel/learnrhash).

Based on my experience, I offer the following best-practices advice. First, check package builds on multiple platforms. I have students on PCs, Macs, and Chromebooks. Issues and problems differ by platform. I use the rhub package (Csárdi and Salmon 2019) to check my package build on various platforms. The check_for_cran command provides a test on windows and linux and check_on_macos tests on the Mac. If at all possible, try checking on a second machine with a clean install. Make sure that students with previous installs update to the latest versions of R and RStudio.

10. CONCLUSION

The *learnr* package provides instructors with tools that allow for development of interactive online instruction. Through integration of video, executable code, quizzes, and shiny apps, courses move beyond simple videos to a more engaging format that allows students the opportunity to practice what they learned from viewing a video.

REFERENCES

Csárdi, G., and Salmon, M. (2019), rhub: Connect to "R-hub."

Guo, P. J., Kim, J., and Rubin, R. (2014), "How video production affects student engagement: an empirical study of MOOC videos," in Proceedings of the first ACM conference on Learning @ scale conference, L@S '14, New York, NY, USA: Association for Computing Machinery, pp. 41–50. https://doi.org/10.1145/2556325.2566239.

Kornell, N., and Bjork, R. A. (2007), "The promise and perils of self-regulated study," Psychonomic Bulletin & Review, 14, 219–224. https://doi.org/10.3758/BF03194055.
RStudio Team (2020), RStudio: Integrated Development Environment for R, Boston, MA: RStudio, PBC.

Sana, F., Yan, V. X., Clark, C. M., Bjork, E. L., and Bjork, R. A. (2021), "Improving conceptual learning via pretests," Journal of Experimental Psychology: Applied, American Psychological Association, 27, 228–236. https://doi.org/10.1037/xap0000322.

Schloerke, B., Allaire, J. J., and Borges, B. (2020), learnr: Interactive Tutorials for R.

Warnock, S. (2013), "Frequent, Low-Stakes Grading: Assessment for Communication, Confidence," Faculty Focus | Higher Ed Teaching & Learning.

Wickham, H., Hester, J., and Chang, W. (2020), devtools: Tools to Make Developing R Packages Easier.