



## Article

## Research Article

JDSSI of M-SCHOLAR  
NO.00024 (A-2024)

# Motivating Mature Learners to Adopt Reproducible Workflows in a 60-minute Hybrid Workshop: A Curriculum Design Challenge

Lisa Y.W. Tang <sup>1\*</sup>, Chung Wai Tang <sup>2</sup><sup>1</sup> Northeastern University, Boston, USA; Simon Fraser University, Burnaby, Canada; University of British Columbia, Vancouver, Canada<sup>2</sup> University of Nottingham, Nottingham, UKReceived: August 15, 2024  
Accepted: October 17, 2024  
Published: November 30, 2024**Principal Author(s):**

Lisa Y.W. Tang is a STEM educator and senior scientist whose research leverages application of artificial intelligence to drive healthcare innovation and social impact. Chung Wai Tang, a seasoned school administrator and business educator with 15 years of experience, is pursuing his Master's degree to craft innovative curricula that integrate Eastern and Western pedagogies.

**Correspondence email:** \*  
l.tang@northeastern.edu**Copyright:** © 2024 by the author(s). Published by Michelangelo-scholar Publish Ltd.

This article is published under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International (CC BY-NC-ND, version 4.0) license, which permits non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited and not modified in any way.

**Publication Statement:** *Journal of Design Service and Social Innovation* focuses on design research and cultural dissemination, but does not involve any political views or cultural biases.

Assistant Editor: Devyn Zhao

**Abstract:** This article presents a novel curriculum design solution for teaching reproducible workflows using Git and Python to mature learners. To accommodate time-constrained learners, our hybrid workshop (offering in-person and remote participation) condenses multiple topics into a one-hour format. The workshop features an engaging *Slido* deck, a live demonstration of *Streamlit* for rapid web app deployment, and a take-home handout summarizing key lessons. The workshop was attended by 46 participants (6 in-person). Participants' feedback collected throughout the workshop suggest high engagement through *Slido* activities and lively discussions. Observations suggest that automation of *Slido* activities and extending the workshop to two hours could enhance future executions. This article discusses lessons learned and provides recommendations for adapting the workshop to diverse audiences.

**Keywords:** Reproducibility; Research transparency; Workflows; Python; *Streamlit*; *Slido*; Colab; GitHub<https://doi.org/10.59528/ms.jdssi2024.1130a24>

## Introduction

In evidence-based research studies, it is crucial that evidence reported in publications are “repeatable” or “reproducible” [1]. In some fields, evidence-based research studies must follow the “FAIR” principles, i.e. Findable, Accessible, Interoperable, and Reproducible [1, 2].

A key component to implementing reproducible studies is the adoption of a version control system (VCS) where researchers create and track a time-stamped version of a piece of data analysis script. The careful tracking of versions is an important propeller of rapid software development and functional and efficient collaborations. To illustrate, with VCS, member A of a team may work with version 1 of a code base (i.e. foundational code) while team member B could choose to create version 2 of the same code base to implement refinement features. Without a reliable system to track these independently written versions, member B must wait for member A to complete before embarking on their tasks. With VCS, the two members could work independently and resolve the differences in the two versions later, when the differences would be highlighted line-by-line by the VCS.

Depending on the specific needs of a subfield, researchers may also use VCS or other alternative platforms to archive and share mathematical models that were developed by an accompanying piece of code that got archived in the same location, often referred to as “repository” or “repo.”

One popular VCS is Git [2]; it is popular because it also integrates “journaling” mechanisms into analysis workflows [1, 3, 4] where “git messages” make up journal entries that would document how a piece of code got revised as well as the rationales behind each code revision. With these journal entries, changes made to code can be easily reviewed, as the rationales are documented at the time of each entry submission. Collectively, these detailed journal entries help ensure different workflows of a research study remain transparent throughout so any external auditors or new team members could review in the near or distant future.

A related merit that may come along with the adoption of a VCS is explainability, which is to document decisions made during the study design process that have led to successes and failures. Without these repositories and accompanying documentations and other version control features, these important decisions and subsequent consequences often go unreported in published articles, due to the side-effects of the review-rebuttal interactions in most editorial processes [1, 5].

Reproducible software stacks such as frameworks based on Python and R programming languages are one popular paradigm embraced by multiple scientific research communities [6]. This article presents a curriculum design challenge and the developed solution for a hybrid workshop that was aimed at encouraging the adoption of practices to enable reproducible research at a private academic institution. We do so through demonstration of multiple technologies integrated into a single theme: deploying a web application using Git and Python.

The workshop was designed for mature attendees recruited from a private academic institution who were engaged in part-time post graduate studies and/or part-time employment at the institution. Note that the curriculum design of this workshop is especially challenging due to our ambitious overarching goal of covering too much over too little time [7, 8, 9], as well as varied psychological needs of the mature learners [10, 11].

In the coming sections, we provide the constraints demanded by the target audience. To help readers understand the problem statement, we first present the technical tools to be covered in the workshop to illustrate how they are connected and how they facilitate reproducible workflows. We then present the developed curriculum and workshop schedule, the rationale behind different design decisions. We will also present findings of the dry-run and actual event. In the conclusion, we discuss strengths and limitations of our approach, and propose different changes that could be explored by future educators. Throughout the remaining text, we will use “web application” and “web app” interchangeably.

# Background

## Review of Technical Tools Discussed in This Article

- *GitHub* is a free platform that allows users to archive their work in a manner that tracks history [2]. It is best leveraged when multiple users collaborate on a project where subset of team members could divide work so that an approach known as “divide-and-conquer” is used to ensure that conflicts between subgroups can be more easily resolved.
- *GitLab* is another platform alternative to GitHub; the two have very similar names albeit owned by two independent companies. Both platforms use a more general technical framework known as “Git” that would require users to know a set of commands when engaged in different phase of a project. The two choices are inherently different with select features appearing only on one of the two choices at times. More domain-specific experience is needed before one could make informed decisions on the choice.
- *Python* is currently one of the most popular programming languages among the scientific research communities across different areas spanning from environmental science, physics, and machine learning/ deep learning engineering for software developed leveraging artificial intelligence [12].
- *Colab* notebook is one subcomponent of the Google suite that can be launched through an Internet browser upon sign-in. Colab Jupyter notebooks allow users to interactively observe the result of code snippets [13]. Figures such as pie charts and scatter plots may be generated on-demand.
- *Streamlit* is the name of a Python library that allows programmers to prototype an interactive application using “lightweight” function calls [14]. Unlike other alternatives such as Shiny, *Streamlit* frees the programmers the need to plan and implement separate code that emulates “UI” and “server” components, which could be daunting for users new to dashboard creations.
- *Streamlit.io* is a cloud platform that permits users to deploy web applications with a simple user interface. The steps on how to deploy an app are presented in a later section.
- *Markdown*, *Hyper-Text Markup Language* (HTML) and *Latex* are languages that relate to (web and print) publishing; examples of the markdown and HTML are given in Section 4.5.
- *Slido* is a platform that allows users to embed interactive widgets in their slide presentations. This technology requires participants to have access to a web browser launched on a desktop computer or mobile device. Participants review their devices for prompts to questions raised during the workshop. Other prompts include survey prompts where users could type-in non-structured text and multiple-choice questions where users could click on a radio button that correspond to the most appropriate answer.
- *R* is another popular programming language with its own set of grammar (known as “syntax”).
- *Tableau* is a proprietary software that is often regarded as industry-standards for the creation of data dashboards. Being proprietary and specifically designed for dashboard

creations, this software may be regarded as less versatile than Python and R.

- *SAS* and *STATA* are popular software tools for more conventional approaches to statistical analyses.

We now explain how these technologies work together by revisiting the previous scenario with members A and B. Let us suppose their project goal is to create an interactive web app. The creation of apps for display on the Internet browser often requires some basic knowledge in HTML. Tableau is one proprietary software for building such apps without demanding knowledge in HTML and other programming languages. Alternatively, free software for web app development includes using open-source programming languages such as R and Python.

The use of Python's *Streamlit* library frees Python developers from learning HTML. The equivalent for R developers is yet to come; those who use R's Shiny library would need to have some working knowledge of HTML. Lastly, for programmers who have training in (bio)statistics and/or data science, SAS, STATA, and R are similar programming languages, albeit SAS and STAT are also proprietary.

Regardless of which programming language elected and what outcome to achieve (i.e. conducting statistical analysis, or creating a data dashboard that shows non-interactive graphs, or a web app that allows users to point-and-click to zoom onto a graph, or sort a table by a column), the adoption of a reproducible workflow helps ensure that results to be generated are reliable and deterministic, and that the entire development is tracked and documented.

As readers may have noticed, we do not teach *Slido* in our workshop but exposes the attendees to its utility for running interactive sessions and making presentations more engaging. At the start of the workshop, participants would first be directed to a *Slido* webpage via a QR code shown on screen, participants would then use *Slido* to answer questions being raised on various topics as introduced earlier.

We next present the background on reproducible workflows and reproducible research/ studies.

## Tools Needed for Reproducible Research vs. Reproducible Workflows

In lay terms, a reproducible research study is one that can be repeated by another independent group of researchers who would be generating results using the study materials that are identical or different from study *I*, and subsequently be able to observe new results that are like those reported in study *I* [7, 15]. Note that reproducibility is different from replicability, which could involve following the execution plan of study *I* on a completely different cohort (see details in [16]).

A reproducible study relies on a carefully documented set of workflows. A well-prepared documentation of a reproducible workflow ensures that the end users can follow the instructions without doubts as all needed information is provided clearly in the document [1].

Note that reproducible workflows may come in many forms; examples include a standard operating procedure for data collection, for generation of a list of invitees to an online survey,

for enrolling participants to a study, etc. The core of reproducibility is that the workflows are well documented in such a way that they can be reviewed and followed by team members consistently throughout the lifespan of a research project, without needing further clarifications from the original writers of the documentations.

Another form of workflow that we focus on in this article consists of the setup of an online repository (using GitHub) that contains programming scripts and detailed listing of required software packages. The latter is saved in a text file named `requirements.txt`. In essence, this file allows programmers to know which exact packages are needed to recreate a web app as well as which version one would need. Here are contents of an example of such a file:

---

```
numpy>=1.26.4
streamlit==1.31.1
streamlit-folium==0.18.0
pyarrow==16.1.0
st_pages==0.4.5
```

---

The example above advises other researchers that a total of five Python packages are needed, and the version number of each package is noted, e.g. one must use version number greater or equal to 1.26.4 for a popular Python package named called *numpy*. Note the use of two equal signs when the version number must match exactly.

The creation of a repository is one key ingredient to the exact creation and recreation of software pipelines and/or data analyses frameworks. To captivate audience's interest, we propose in this article a curriculum that focuses on the task of creating a web app. The choice was determined based on learners' profiles (to be explained in Section 2.3). For advanced programmers, alternative tasks for demonstration may include the (re)creation of machine learning cross-validation pipelines, and/or deployment of fitted models for predictive tasks. **Table 1** provides a set of high-level instructions for our target task.

**Table 1.** An example reproducible workflow that involves code archival and deployment of web apps .(drawn by the authors)

---

- 1) Prepare Python programming script(s).
  - 2) Archive the script(s) on a GitHub repository, along with a file named `requirements.txt` that serves to describe the computing environments required to be rebuilt on the Streamlit.io server.
  - 3) Create a user account for access to Streamlit.io.
  - 4) Login to Streamlit.io and select "Create app."
  - 5) When prompted, choose to specify the URL address of the GitHub repository that was created in step 1.
  - 6) Adjust the default settings (e.g. permitting the app to be made available to the public, or only to owners of email recipients whom would receive email invitations to the private version of the app).
  - 7) Allow time for the Streamlit.io platform to reproduce the compute environments using the specifications captured by the GitHub repository.
-

**Table 2.** Interactive and networking opportunities were top requests. (drawn by the authors)

	Staff				Trainee				
	#1	#2	#3	#4	#1	#2	#3	#4	#5
R session	√					√	√		
Python	√	√	√						
Statistics					√				
Writing tips					√				
Networking				√				√	
Interactive									√

## Understanding Learners’ Profiles

To draft a set of guidelines to follow when making fine-grained decisions (e.g. deciding on the difficulty of content shown), we first made qualitative observations about the learners’ immediate and long-term work environment. Subsequently, short informal interviews were conducted with learners recruited from past events. Based on the interviews, we found that half of the participants advocated for more hands-on training in Python, while half also advocated for interactive lessons and opportunities for social activities. We also projected views of learners’ profiles as summarized in **Table 2**. Note that the opportunity for networking was regarded important. While these are only projected views (because the interviewees were recruited from past events and thus do not represent the characteristics of the target audience), they helped us prioritize the contents and help steered time-allocations.

To analyze common patterns in the participants’ profiles, we also grouped the projected attendees by two categories: staff and students. Students generally had more autonomy than staff. Based on analysis of the informal interviews, we hypothesized that staff may have work obligations during the event that would divert their attention towards side tasks that can be executed in a remote setting but more difficult to execute when the workshop delivery mode is restricted to in-person only. Due to the work-arrangements promoted since COVID19 remote-work policies, we could not demand in-person participation. At the private academic institution where the workshop was designed initially for, it was projected that only one-third of registrations would end up attending the event. Due to this reason (and other administrative concerns at this institution), pre-registrations were not setup.

## Design Constraints & Challenges

We aim to design a one-hour hybrid workshop curriculum to be hosted under the following design challenges (C#1-6):

- C1: One hour to cover the importance of reproducibility and hands-on experience with Git and Python;

- C2: Distribution of learners' profiles is unknown at the time of curriculum development;
- C3: Face-to-face interactions to gauge learner's attention span is not an option for all participants;
- C4: Learners' varied attention spans, e.g. some may be multitasking with lunch-consumption and/or responding to text messages from other colleagues outside of the workshop;
- C5: Learner's varied interests and skill levels, e.g. may have no formal training in programming in any language, i.e. no experience in Python nor R;
- C6: Most participants employ graphical user interface in day-to-day tasks and are not familiar with command line interfaces (CLI), while experienced Git users frequently advocated the use of CLI.

## Problem Statement Defined

After accounting for the challenges listed in the introduction, we set two objectives as workshop facilitators:

- O1: Cultivate a sense of confidence in learners throughout the workshop; and
- O2: Sustain active use of the illustrated technologies after the workshop.

Due to the one-hour restriction, we aim for a curriculum whereby all participants would have gained a sense of competency in Python coding during the workshop, regardless of their programming skills level prior to the workshop.

As we elaborate in the upcoming sections, the workshop draws audience's interest through a live demonstration that teaches learners in less than ten minutes how to repeat the creation of a web app using *Streamlit* and other Python packages using a repository made accessible on GitHub. This specific combination of technologies was chosen to tailor to the projected profiles of the target audience, which, in essence, are those with limited practical experience in Git and/or Python.

## Methods

### Overview of the Proposed Approach

On a high level, we divided the one-hour hybrid workshop into four parts.

The first part involved learners actively learning and participating through a *Slido* deck that interlaced quiz-like prompts of varying difficulty levels. In *Slido*, we activated the "scoring chart" feature so that the participants could see the names of participants with the top five scores right after each question.

The second part of the workshop was designed to sustain interest after the workshop, which may be difficult due to challenges C2, C4, and C5. As a remedy, we present a state-of-the-art web app deployment approach based on Python's *Streamlit* package as the highlight of the workshop that may be summarized with a summary handout.

## Adopted Workshop Schedule

P0: Introduction [3 min]

P1: *Slido* deck with some demo on Colab Jupyter notebook [20 min]

P2: Short introduction on the technologies described in Section I [10 min]

P3: Short presentation entitled “Deploying a web app with reproducible workflows under ten-minutes with seven steps” [15 min]

P4: Live demonstration on *Streamlit* to illustrate at greater depths the steps of creating a web-app [10 min]

## Design Rationales of the Curriculum and Workshop Schedule

As similarly advocated by other educational workshops [17], we elected the use of Google Colab notebook to achieve the first objective (O1) due to Colab’s ease of use and high accessibility. As alluded to in the Introduction section, Colab is a cloud-based platform that the public could gain access for free after a sign-up process. Upon sign-up, Colab Jupyter notebooks can be launched on most web browsers within seconds. Colab notebook permits both Python and R programming. We also hypothesized that the ability to launch Colab sessions for quick calculations and data analysis queries will be appreciated by workshop participants, which help to solicit their interest to practice Python programming after the workshop.

When reviewing Python syntax, Colab notebook would be used to demonstrate on-demand calculations, such as showing the output of code blocks containing the following code snippets:

```
a = 2*2           # a stores value of 4
print(a*2 + 1)   # Prints out 9
b = '2'*5        # b stores value of '22222'
print('Hello!' + " " + b) # Prints out 'Hello! 22222'
```

In Python (and R), a sequence of alpha-numeric characters is called a “string” and a variable of string type can be marked by a single or double quotation marks as shown in line 4 above [18]. In short, the above examples would allow us to explain to audience how line 1 and line 2 differ in terms of data types, as hinted in the comments marked after #. They also illustrate how the plus sign (+) and multiplication sign (\*) are “overloaded,” which is a term to refer to how their utilities differ depending on the variables they operate on. The \* sign in line 1 indicates a mathematical operation for multiplication (two multiplies with two) while in line 2 indicates a repeat operation (the string '2' is repeated for five times). The + sign in line 3 indicates a summation while in line 4 indicates a “join” operation, i.e. it joins three string variables, namely, 'Hello!', " ", and '22222'.



## Dry-run Sessions

Prior to the actual workshop, we conducted two separate sessions of dry-runs to help guide us revisions needed for the workshop agenda. After few rounds of gentle invitations, we successfully recruited four and one volunteers for the first and second dry-run, respectively.

In these dry run sessions, we explored the following questions:

- How many of these questions should focus on reproducibility and how many should focus on specific technologies?
- How to best engage the participants in a comfortable way?
- Which candidate activities should be included in the final agenda?

## Results

### Participants at the Dry-runs

In the first dry-run, two of the recruited volunteers knew each other, while the remaining two did not know any attending volunteers. The diversity of backgrounds included one R user, one Tableau user, and two project administrators.

After self-selection, one R user got paired with one project administrator.

In the second dry-run, the only participant was a SAS user. All recruited volunteers have had no prior experience in Git, Python, *Streamlit*. The second dry-run was done in a slightly different manner but remained useful in that it helped to confirm decisions made after the first dry-run.

### Findings from the Dry-run

Code-along is an extension of live-coding [19] where a lecturer or course instructor thinks out loud while writing code on the screen so that participants could learn from the instructor's thought processes. Code-along permits participants to inject questions at any time, which is less formal than live-coding demonstrations used in conventional lectures.

A variant of code-along is paired-coding whereby peers review a notebook together. In the first dry-run session, we explored this approach where we drafted sample notebooks for audience. In these notebooks, we embedded coding questions where pairs of participants would try to answer on their own. The questions were themed under a domain-specific topic (e.g. geo-mapping of spatial data, or natural language processing). All solutions to each question would be reviewed collectively near at the end of the code-along session.

Results of the dry-run suggest this approach would not work due to insufficient warm-up times allotted in the beginning. As a result, we observed that paired participants felt uncomfortable with each other and were not able to engage in lively conversations pertaining to the topic.

After informal discussions with the volunteers from the dry-runs, we elected to spend equal portions of Part 1 on Python syntax, GitHub, and concepts around reproducible research/workflows.

**Table 3.** Example design guidelines. (drawn by the authors)

---

Python and R: the two programming languages are promoted as supplementary tools rather than competing methods.

---

We prioritize workshop objectives of soliciting participants' long-term interest over success of grasping new concept that are not critical self-learning in the long-term.

---

Graphical user interface should have higher precedence than command line interface.

---

**Table 4.** Example of design decisions. (drawn by the authors)

---

Creation of Git logs that are systematic and effective [1] is considered too advanced; we omitted the introduction in our workshop because the ability to generate logs are not critical for novice git users. On the other hand, its introduction will likely slow down pace of the workshop and/or render content difficult to follow.

---

The original idea of including paired-coding was dropped due to difficulty of ensuring high engagement levels for all participants.

---

We elected to deploy web apps remotely via account creation of the Streamlit.io.

---

**Table 3** and **4** present examples of the developed guidelines and corresponding decisions made after integrating observations from the dry-runs.

## Participants at the Actual Workshop

According to the statistics collected, about forty participants joined online at the start of the hour. Of the six in-person participants, at least two were dominant R users, another two were STATA users, and another one was a SAS user. As registration was not required, the profiles of most online participants in the actual workshop were not traceable. Based on prior knowledge (from informal interviews), at least two online and one in-person participants had experience in Python programming.

## Execution of the Actual Workshop and Observations Therein

In the introduction formulated as an ice-breaker, we advised participants the use of *Slido* and requested them to visit a "URL where the *Slido* deck was hosted." To cultivate a relaxed atmosphere, we advised all participants the adoption of a nickname that would keep their identity anonymous. To illustrate, we presented a snapshot where a nickname "Albert Eistein" could be used. This solicited a few laughs, based on observations on the in-person participants.

The live demonstration in Parts 3 and 4 included detailed explanations of the following components:

- Code structure on GitHub repository

- requirements.txt: A file that lists the software libraries required by the demo web app as introduced in Section 2.2;
- Logging into the streamlit.io web interface and introduction of the widgets and features
- Introduction of CodeSpace (which is a code editor)

## Findings of the Actual Workshop

Out of the 46 participants who joined at the start of the hour, 32 registered on our *Slido* deck (two out of the six in-person participants elected to not use *Slido*).

We next report a summary of the participants' responses as recorded on *Slido*.

In terms of technical experience, 67% of them have worked with Markdown before the workshop (50% with Excel scripting; 33% with GitHub; 17% with Latex publishing; 28% with Power Bi; 89% with R programming; 50% with SAS scripting; 33% Python programming).

When trying to gauge audience's general interest in *Streamlit* during the ice-breaker, 12% of the respondents chose the response of "Yes, I got *Streamlit* account for the purpose of this workshop." Other topics received the following percentages:

- Reproducibility research: 54%
- Python 101: 50%
- GitHub pages: 50%
- Prototyping web apps: 38%

When assessing knowledge level on a markdown snippet, the following was shown on the screen and participants were asked to identify what could be recognized:

---

```
# title
## subheading
[x] bananas
[x] oranges
[ ] <br> [apples] (a)
```

---

For this question, 100% of respondents correctly answered that this code snippet contains no typos; 69% correctly answered that markdown snippet was shown; 54% correctly answered that the code contains HTML and a checklist; 23% correctly answered that the code contains "content."

To stimulate lively discussions, we also designed and raised True/False questions that were purposely debatable; examples are given in **Table 5**. The last example was particularly notable for its debatability, largely due to the reference to a "blue pen." This enabled the author to draw on historical accounts of the need to transmit documents via fax machines, which often required the use of black ink for optimal legibility of the transmitted printouts.

**Table 5.** Example True/False questions that were designed to stimulate lively conversations. (drawn by the authors)

---

In R, variables can start with a period, e.g. .patientIDs. Conversely, in Python, the period should not be arbitrarily used because it is dedicated operator.

---

Regarding this line: stats::filter() In both Python and R, “stats” refers to a namespace that helps us resolve identical function names defined in different packages. Use of namespace is a good practice that enhances repeatability.

---

Researchers may submit a completed checklist that ensures reproducibility of published results without being about having completed t tasks noted on the checklist.

---

Reproducibility involves carefully documenting every detail of your work environment, and always using a blue pen.

---

## Participants’ Responses Collected through *Slido*

After the administration of the True/False questions, *Slido* was also used to solicit and participants’ open questions which were subsequently answered by the author before proceeding to Part Three of the workshop. Example of these questions include:

- “How could we potentially use large datasets with these tools?”
- “What are the limitations using Python?”
- “Why/ when to use Python over R?”
- “What are the advantages of Python over R?”

The next set of slides involved presenting more complicated concepts in Python that were potentially new to many of the participants. Unfortunately, not all statistics could be collected due to a glitch stem from using a free version of the *Slido* tool. The only statistics that we could track came from the first question: 38% of the respondents answered the first knowledge-assessment correctly.

The next part of the workshop was facilitated by a slide deck to demonstrate how one could use a handout to follow the steps of deploying a *Streamlit* web app. As part of post-lesson evaluation, this part ended with a poll-like question shown on screen asking how confident the participants felt in their ability to deploy a *Streamlit* web app by themselves today (after the workshop). On a scale of 7, 25% of respondents rated 6 while 38% rated 2. Equal percentages (13%) of responses rated their confidence levels at 3, 4, and 5.

A quick recap of the steps were illustrated before closing the workshop. After the recap, at which point only a few minutes remain before the hybrid workshop must end, one last question was raised to the audience: “How might web apps be useful in your daily work?”. Many online participants have left the workshop by this time. Nevertheless, this question solicited five responses recorded on *Slido* that could be grouped into three distinct tasks/themes:

- “For data storage and analysis.”
- “For creation of a dashboard.”

- “To offer flexible and interactive reporting options.”

The listing of the above responses which were made anonymously was verbally summarized by the workshop host as an informal lesson summary. These answers suggest that the central theme of the workshop was understood by the audience [8].

One “Thank you” was received on the online channel before the channel was automatically shut off. Two of the in-person participants stayed behind to provide informal feedback on the format. As the participants were colleagues of the workshop host, it would not be feasible to interpret the verbal comments received. On the other hand, the interactions between the host and their colleagues helped team-building as the idea of a follow-up meeting for deeper discussions was raised.

## Discussions

### Strengths & Limitations

One of the major drawbacks of running a hybrid workshop is the inability for the presenter to attend to the needs of both groups of audience at the same frequency. While conscious efforts were made to attend to online participants on a regular interval, a feedback-loop was only possible with the in-person participants who provided continuous feedback in forms of facial and body expressions.

Another limiting factor is the choice on Google Colab, as some participants expressed reluctance in supporting/using Google products in general. Indeed, during the workshop, another coding platform called *syzygy.ca* [20] was examined as an alternative to learn Python/ R programming. Based on *Slido* responses, only 21% of respondents have access to this option as it is currently being administered by public academic institutions and thus require affiliations with these institutions either through course enrolment and/or employment. Cloud-based resources such as Colab remain to be attractive choice to the public since no special computer hardware is required and generally accessible with minimal setup costs. Again, we acknowledge that the requirement of Wi-Fi can be highly restrictive for remote communities. Local partnerships with communication companies could be explored to help remote communities on this front in the future.

On a broader level, our workshop taught content that demanded constant access to the Internet, albeit the *Streamlit* package itself does not enforce the condition that all applications must be deployed on the server. Indeed, applications can be deployed locally. This may be an acceptable tradeoff given that local deployment required more advanced experience in Python than the remote deployment alternative.

The strengths of our curriculum design choices may be articulated with pedagogical perspectives as follows.

The use of *Slido* decreased the disparity between online and in-person participants by bringing both audience groups to the same attention mechanism. It also helped streamline the hybrid workshop by offering inclusive environments in the same manner to everyone who chose to join the *Slido* activities. Overall, *Slido* enabled us 1) to conduct initial

assessment of learners' skills and knowledge that would help workshop host determine the pace of the workshop; 2) to present new concepts in a quick and stimulating fashion by providing answers and explanation after each question, followed by the presentation of the "scoring chart."

In education, pedagogical theories provide a basis for understanding the learning process, which is necessary for determining suitable approaches to support learning. One school in these theories focus on behavioural characteristics. For instance, a positive reinforcer (or a reward) may encourage a desired behavior whilst a negative reinforcer (or a punishment) may discourage an undesirable behavior [21].

Using reinforcers such as praising participants for their good performance [22], or introducing a reward system, may lead to behavioral change [23]. In our workshop, a "scoring chart" being shown after each *Slido* activity served to elicit behavioural changes in both directions. We could not compare the degrees of positive and negative impact solicited during the workshop but one may conjecture that an incorrect answer served to increase participants' attention levels while correct answers served to excite participants and maintain their interest levels. Future workshop executions could derive strategies to count the number of correct responses a given participant gave relative to the number of total responses gave to examine the extent of positive changes solicited by the "scoring chart." Nevertheless, readers should keep in mind that "correctness" as a measurement of positive learning experience may not always be applicable.

Related to this discussion is another school of theories collectively known as constructivism, which sees learning as an active process of knowledge construction. The tenet of constructivism is that learners create new understanding from prior knowledge and past experiences. According to constructivists, no two learners have the same interpretation of reality since each learner possesses unique knowledge and experiences. This view maintains that the learning process is based on exchanging ideas through two dimensions: the cognitive dimension recognizes that the mind receives information and constructs its own vision of reality; the social dimension recognizes that knowledge is co-produced by collaboration. In our case, through the use of *Slido* where questions were raised and participants were asked to develop their own answers independently, followed by integration of other's answers has helped to engage learners' thought processes on both cognitive and social dimensions.

## Future Extensions

Our curriculum design solution was aimed at addressing desirable goals of being short, practical, and interactive while accounting for the set of learning conditions when making low-level design decisions. Below, we suggest few extensions for future adopters:

- An assessment activity could be added before the *Streamlit* demo whereby all participants would provide response on a shared slide deck that would mimic a virtual whiteboard on which participants are invited to type in their answers to prompts such as:
  - a. "Shown on your slide is a raw text. After reading over them, could you use markdown to present this data?"

- b. “Type a Python command that would print a message that tells us your favourite book or movie titles.”
  - c. “How would you expand this Python code so that titles are capitalized?”
- If the workshop length can be increased to two hours, we recommend the inclusion of paired programming sessions executed in two to three parallel break-out sessions on domain-specific themes (number of break-out sessions depend on the number of in-person versus online participants). Example code-along session titles may include:
    - a. “Dashboards with *Streamlit*’s multipage package;”
    - b. “Interactive maps via *Streamlit* and folium”
  - The agenda and curriculum can be improvised for a different set of technology-combination, e.g.:
    - a. R programming language in place of Python
    - b. Posit cloud in place of GitHub/*Streamlit*
    - c. R Shiny in place of *Streamlit*.

The paired programming sessions may be collectively moved to the second hour as a separate (voluntary) exercise and/or followed by social-and-mingle event that would serve as the “bookend” for the one-hour workshop [8]: review the problem and needs at the start of the second hour, review the solutions learned in the middle, and reflect on what worked and did not at the end. In our context, such a bookend would allow learners to recall the purposes of reproducibility workflows, recall the steps of creating the workflow together with their peer, and leaving the workshop with some senses of accomplishment and team spirits.

Lastly, this event would also serve to meet the learners’ psychological and social needs: having the opportunity to talk one-on-one with a peer and to share their emotions with their peers (e.g. feeling frustrated and/or lost that the materials were presented too quickly), as well as the autonomy to explore topics beyond the workshop topics (e.g. converse on non-academic topics just to learn about each other).

Before ending this section, we highlight two new use cases of technologies in education. Firstly, *Streamlit* has been used as a portal creation for an online self-paced learning program as demonstrated in recent work [24]. Secondly, the use of chatbot as an engagement tool in classes has been proposed [25, 26]. The latter idea does raise concerns in terms of privacy and data security. The authors [26] end with a cautionary note that the “human touch remains indispensable.”

## Conclusion

We have designed, implemented, and evaluated a curriculum for a hybrid workshop that was aimed to teach and motivate reproducible workflows and reproducible research. We anticipate that workshop participants from all fields will continue to advocate for a hands-on, interactive learning experience. In our case, with the use of a *Slido* deck, we

were consciously guided to create course content in a manner that would cultivate a friendly and inclusive learning environment.

**Author Contributions:**

Lisa conceived, developed, and held the one-hour workshop curriculum. She also conducted the data analysis and led the manuscript preparation. Chung Wai contributed theoretical insights and pedagogical commentary in the Discussions. Both reviewed and approved the final revision.

**Funding:**

Not applicable.

**Acknowledgements:**

The authors sincerely thank all workshop participants and dry-run volunteers for their valuable feedback and interest to join the one-hour hybrid workshop. We also thank Tang Kim Chuen and Tong Tsui Shan for their immense support as we navigate through our challenging journeys in education.

**Conflicts of Interest:**

The authors declare that they have no conflicts of interest related to this research.

## References

- 1 Samuel J. Huskey, "Committing to reproducibility and explainability: using git as a research journal," *International Journal of Digital Humanities* 6, no. 1 (2024): 9-21. [[CrossRef](#)]
- 2 Janick Weberpals, and Shirley V. Wang, "The FAIRification of research in real - world evidence: A practical introduction to reproducible analytic workflows using Git and R," *Pharmacoepidemiology and drug safety* 33, no. 1 (2024): e5740. [[CrossRef](#)]
- 3 Filippo Lanubile, Silverio Martínez-Fernández, and Luigi Quaranta, "Teaching MLOps in higher education through project-based learning," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 95-100, IEEE, 2023. [[CrossRef](#)]
- 4 Aaron Peikert, Caspar J. Van Lissa, and Andreas M. Brandmaier, "Reproducible research in R: A tutorial on how to do the same thing more than once," *Psych* 3, no. 4 (2021): 836-867. [[CrossRef](#)]
- 5 Mine Dogucu, "Reproducibility in the Classroom," *Annual Review of Statistics and Its Application* 12 (2024). [[CrossRef](#)]
- 6 Alexander S. Ford, Brian D. Weitzner, and Christopher D. Bahl, "Integration of the Rosetta suite with the python software stack via reproducible packaging and core programming interfaces for distributed simulation," *Protein Science* 29, no. 1 (2020): 43-51. [[CrossRef](#)]
- 7 Richard Ball, "'Yes We Can!': A Practical Approach to Teaching Reproducibility to Undergraduates," (2023). [[CrossRef](#)]
- 8 Judith V. Boettcher, and Rita-Marie Conrad, *The online teaching survival guide: Simple and practical pedagogical tips* (John Wiley & Sons, 2021).



- 9 K. Jarrod Millman, Matthew Brett, Ross Barnowski et al., "Teaching computational reproducibility for neuroimaging," *Frontiers in Neuroscience* 12 (2018): 727. [[CrossRef](#)]
- 10 Kelly McKenna, Kalpana Gupta, Leann Kaiser et al., "Blended learning: Balancing the best of both worlds for adult learners," *Adult Learning* 31, no. 4 (2020): 139-149. [[CrossRef](#)]
- 11 Yafei Shi, Qi Cheng, Yantao Wei et al., "Understanding the effect of video conferencing learning environments on students' engagement: The role of basic psychological needs," *Journal of Computer Assisted Learning* 40, no. 1 (2024): 288-305. [[CrossRef](#)]
- 12 Yun Peng, Ruida Hu, Ruke Wang et al., "Less is More? An Empirical Study on Configuration Issues in Python PyPI Ecosystem," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1-12, 2024. [[CrossRef](#)]
- 13 Ricardo Ferreira, Michael Canesche, Peter Jamieson et al., "Examples and tutorials on using Google Colab and Gradio to create online interactive student - learning modules," *Computer Applications in Engineering Education* (2024): e22729. [[CrossRef](#)]
- 14 Yaganteeswarudu Akkem, Saroj Kumar Biswas, and Aruna Varanasi, "Streamlit-based enhancing crop recommendation systems with advanced explainable artificial intelligence for smart farming," *Neural Computing and Applications* 36, no. 32 (2024): 20011-20025. [[CrossRef](#)]
- 15 Benjamin A. Antunes, and David RC Hill, "Reproducibility, Replicability, and Repeatability: A survey of reproducible research with a focus on high performance computing," *arXiv preprint arXiv:2402.07530* (2024). [[CrossRef](#)]
- 16 Steven N. Goodman, Daniele Fanelli, and John PA Ioannidis, "What does research reproducibility mean?," *Science translational medicine* 8, no. 341 (2016): 341ps12-341ps12. [[CrossRef](#)]
- 17 Elena Miller, Katy Shaw, and Zachary Dodds, "Choosing Our Computing Birthplace: VSCode vs Colab as GenEd IDEs," *Journal of Computing Sciences in Colleges* 39, no. 1 (2023): 103-111.
- 18 Matthew J. Smith, Mohammad A. Mansournia, Camille Maringe et al., "Introduction to computational causal inference using reproducible Stata, R, and Python code: a tutorial," *Statistics in medicine* 41, no. 2 (2022): 407-432. [[CrossRef](#)]
- 19 Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson et al., "Role of live-coding in learning introductory programming," in *Proceedings of the 18th koli calling international conference on computing education research*, pp. 1-8, 2018.
- 20 Fani Boukouvala, Alexander Dowling, Jonathan Verrett et al., "Computational notebooks in chemical engineering curricula," *Chemical engineering education* 54, no. 3 (2020).
- 21 Burrhus F. Skinner, "Operant conditioning," *The encyclopedia of education* 7 (1971): 29-33.
- 22 Kevallyn R. Drake, and Gena Nelson, "Natural rates of teacher praise in the classroom: A systematic review of observational studies," *Psychology in the Schools* 58, no. 12 (2021): 2404-2424. [[CrossRef](#)]
- 23 Ruth Payne, "Using rewards and sanctions in the classroom: Pupils' perceptions of their own responses to current behaviour management strategies," *Educational Review* 67, no. 4 (2015): 483-504. [[CrossRef](#)]
- 24 Hanna Siebert, Marian Himstedt, and Mattias Heinrich, "Learn2Trust: A video and streamlit-based educational programme for AI-based medical image analysis targeted towards medical students," *arXiv preprint arXiv:2208.07314* (2022).
- 25 Chinedu Wilfred Okonkwo, and Abejide Ade-Ibijola, "Python-bot: A chatbot for teaching python programming," *Engineering Letters* 29, no. 1 (2020).
- 26 Pedro Filipe Oliveira, and Paulo Matos, "Introducing a chatbot to the web portal of a higher education institution to enhance student interaction," *Engineering Proceedings* 56, no. 1 (2023): 128. [[CrossRef](#)]