Check for updates

METHOD ARTICLE

REVISED Improving the usability of open health service delivery simulation models using Python and web apps

[version 2; peer review: 3 approved]

Thomas Monks[1,2], Alison Harper[1-3]

[1]University of Exeter Medical School, University of Exeter, Exeter, England, UK
[2]NIHR Applied Research Collaboration South West Peninsula, University of Exeter, Exeter, England, UK
[3]University of Exeter Business School, University of Exeter, Exeter, England, UK

v2 **First published:** 05 Oct 2023, **3**:48
https://doi.org/10.3310/nihropenres.13467.1

Latest published: 15 Dec 2023, **3**:48
https://doi.org/10.3310/nihropenres.13467.2

Open Peer Review

Approval Status ✓ ✓ ✓

	1	2	3
version 2 (revision) 15 Dec 2023		✓ view	✓ view
version 1 05 Oct 2023	✓ view	? view	? view

1. **Geraint Palmer**, Cardiff University, Cardiff, UK
2. **Robert Smith**, The University of Sheffield, Sheffield, UK
3. **Apostolos Kritikos**, Aristotle University of Thessaloniki, Thessaloniki, Greece

Any reports and responses or comments on the article can be found at the end of the article.

Abstract

One aim of Open Science is to increase the accessibility of research. Within health services research that uses discrete-event simulation, Free and Open Source Software (FOSS), such as Python, offers a way for research teams to share their models with other researchers and NHS decision makers.

Although the code for healthcare discrete-event simulation models can be shared alongside publications, it may require specialist skills to use and run. This is a disincentive to researchers adopting Free and Open Source Software and open science practices. Building on work from other health data science disciplines, we propose that web apps offer a user-friendly interface for healthcare models that increase the accessibility of research to the NHS, and researchers from other disciplines. We focus on models coded in Python deployed as streamlit web apps.

To increase uptake of these methods, we provide an approach to structuring discrete-event simulation model code in Python so that models are web app ready. The method is general across discrete-event simulation Python packages, and we include code for both simpy and ciw implementations of a simple urgent care call centre model. We then provide a step-by-step tutorial for linking the model to a streamlit web app interface, to enable other health data science researchers to reproduce and implement our method.

Plain language summary

Simulation models provide a quantitative way for researchers to make predictions about complex health services, for example to assess the effects of changes to patient care pathways. The most common approach used for health services research is discrete-event

simulation. Historically, research has used software that must be purchased and has restrictive licensing. This can make it difficult for other researchers, and NHS staff such as managers and clinicians, to use the model to help with their planning and resourcing decisions.

One aim of Open Science is to increase the accessibility of research. Free and Open Source Software (FOSS) such as Python offers a way for research teams to share their models with other researchers and NHS decision makers. Although the code for simulation models can be shared alongside publications, it may require specialist skills to use and run. Building on work from other health disciplines, we propose that web apps offer a user-friendly interface for healthcare models that increase the accessibility of research. A web app runs in the browser of a computer and allows users to update model parameters, run different experiments, and explore the impact on the health service that is being studied. We focus on a package called streamlit.

To increase uptake of these methods, we provide an approach to structuring model code in Python to enable the model to be easily integrated into streamlit. The method does not depend on a specific discrete-event simulation package. To illustrate this, we developed simulations using two Python packages called simpy and ciw of a simple urgent care call centre. We then provide a step-by-step tutorial for linking the model to the streamlit web app interface. This enables other health data science researchers to reproduce our method for their own simulation models and improve the accessibility and usability of their work.

Keywords
Open Science, Discrete-Event Simulation, Health Services Research, Web Applications, Python, Model Reuse, Reproducibility

Corresponding author: Thomas Monks (t.m.w.monks@exeter.ac.uk)

Author roles: Monks T: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Project Administration, Resources, Software, Supervision, Validation, Writing – Original Draft Preparation, Writing – Review & Editing; **Harper A**: Conceptualization, Investigation, Methodology, Project Administration, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This report is independent research supported by the National Institute for Health and Care Research Applied Research Collaboration South West Peninsula.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2023 Monks T and Harper A. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Monks T and Harper A. **Improving the usability of open health service delivery simulation models using Python and web apps [version 2; peer review: 3 approved]** NIHR Open Research 2023, **3**:48 https://doi.org/10.3310/nihropenres.13467.2

First published: 05 Oct 2023, **3**:48 https://doi.org/10.3310/nihropenres.13467.1

REVISED **Amendments from Version 1**

Since the previous version of the article we have done the following:

- Reformatted all code listings in the article to adhere to the PEP8 style guide using the software 'black' and updated all references to lines of code within a listing.
- Added additional text in 'Code to test the model' section to signpost readers to in depth material on test driven development and unit testing.
- Reformatted Listing 1 into standard pseudo code.
- Deployed an example simulation model interface to streamlit community cloud.
- Split the 'discussion' section into subsections covering 'strengths' and 'limitations'.
- Expanded our discussion of limitations to include the challenges and existing research in relation to private data, model documentation, structuring model code, and customisation of model interfaces.
- Added additional references to key Health Economic Evaluation articles identified by reviewers.
- Expanded and rephrased our discussion of strengths to consider generalisability of our approach to Agent Based Simulation models coded in python.
- Improved the signposting to tutorials covering Health Economic Evaluation using R.
- Renamed the 'Conclusions' section to 'Conclusions and future work'.
- Removed the duplicated 'Plain English Summary' from the article PDF.

The fundamental contribution of the article, the general message and the functionality of our free and open source code remains unchanged.

Any further responses from the reviewers can be found at the end of the article

Introduction

Health service delivery is complex and costly. Computer simulation can be used to provide decision-support for design or re-design of healthcare processes, accounting for the inherent uncertainty associated with service delivery. Models that represent key characteristics or behaviours of a system or process are developed. Experimenting with the model enables exploration of the potential impact of changes to the system without the costs and risk associated with real-world changes[1,2]. In healthcare, discrete-event simulation (DES) is the most common simulation method for modelling[3]. Reviews of the field demonstrate wide-ranging applications in health service delivery, for example evaluating operational performance, improving patient flow, and scheduling services[4–6]. Healthcare DES models are complex research artifacts: they are time-consuming to build, require clinical time and expertise, depend on specialist software and logic, and may be difficult to describe accurately using words and diagrams alone[7].

DES for health service delivery modelling has traditionally been conducted using proprietary software such as Arena, Simul8 and Anylogic[4,5,8]. One reason is that proprietary simulation software is powerful, flexible, and user-friendly. For DES, proprietary licensed software have high costs per year, in the thousands, and restrictive licensing that limits how it can be used. This licensing causes two major issues for Open Science. The first is transparency of model artifacts[9]. If a researcher, or in some cases the NHS, is required to purchase or subscribe to the software, and does not have the means, then they cannot scrutinize how models really work: they cannot reuse the artifact for further research, and in extreme cases are prevented from using a model to support the redesign of NHS services. In some cases software vendors may provide a free-to-use "personal learning edition" of the simulation software. Here licensing allows researchers to inspect model code for their own learning, but legally prohibits use for further research or decision making. To avoid wasting the substantial clinical, and methodological expertise used in building models, new methods to improve the Open Science, transparency and usability of DES models are of critical importance.

Free and Open Source Software (FOSS) for simulation offers an alternative to proprietary software without the licensing restrictions that affect uptake, sharing, and reuse of model code[10,11]. In health service delivery research, computational analyses, and simulation studies can be conducted using FOSS tools built in the popular coding languages of Python, R, and Julia[12–17]. Use of a language such as Python has further benefits for research as models can now be integrated with the wider data science ecosystem for statistical modelling and machine learning. Contemporary approaches are available to publish the exact software and code artifacts used in a study. These provide credit to authors, and a way to cite outputs other than research articles, for example, open science archives (e.g. Zenodo, or the Open Science Framework), model specific archives (e.g. the Peer Models Network[18] and CoMSeS Net[19] and online computational environments such as Code Ocean.

One downside of code based FOSS models is that the switch from commercial software can be challenging. In recent years there has been an effort to reduce this entry barrier with new resources and textbooks aimed at the Operational Research community[11,20,21]. One open challenge is that these models are script-based and do not have have a user-friendly interface. In health service delivery research this may limit the uptake of new research tools for care pathway planning by NHS decision makers. This challenge is faced across all health data science disciplines, and methods are beginning to emerge to enable research to progress using FOSS. For example, in health economics and pharmacology, robust methods have been proposed to build user-friendly interfaces for models developed in R using Shiny, an open source framework for building web applications in R[22,23]. There are few examples of simulation studies for health services delivery that have developed free and open source DES models with a focus on use and reuse, using R or Python[14,24,25]. Several well-maintained DES libraries are available in Python, including simpy[26], salabim[27], and ciw[28], while streamlit provides a relatively simple alternative to Shiny for building web apps in Python.

In this paper we focus on Open Science for health service delivery research and, in particular, Operational Research (OR) studies that use DES to model health care pathways and service delivery using Python. This study is aimed at researchers already familiar with developing healthcare DES models in Python. We detail approaches to enhance open science in healthcare DES by developing a method to organise Python DES models and provide a user friendly interface for model users using streamlit for shareable web apps.

Aims

This paper focuses on healthcare DES models developed using Python. It builds on pioneering work from health economics[22], pharmacology[23], and OR[11,20] to support the community in adopting practices in R and Python FOSS tools to increase transparency and use/reuse of computational research artifacts. To complement existing work integrating Shiny in simulation models, we provide a method to add a streamlit interface to a Python model. We illustrate this with an application in DES using Python frameworks simpy[26] and ciw[28]. We also detail a simple method for structuring Python DES models to enable reuse, extension, and swapping of simulation packages. Before presenting our method we provide an overview of FOSS, Python for computer simulation, and streamlit.

Methods

We developed a method for structuring DES models to enable reuse and linkage to a user-friendly streamlit web app. The method allows users of healthcare models to quickly create and execute one or more experiments with a DES simulation model deployed via a web app using Python. Our work is designed to improve dissemination and implementation of DES research in health services to support patient care.

FOSS and Python have gained significant popularity and use in the UK's NHS in recent years. This transformation has been inspired via communities such as NHS-Python (and NHS-R) as well as NHS-England promoting the use of FOSS tools[29]. The Goldacre Review of research using health data[30] has called for shared, reusable code for data analysis, that minimises inefficient duplication and enables research continuity. FOSS grants the rights for users to adapt and distribute copies however they choose. FOSS software within health data science is provided with an open license.

We chose Python as it is consistently ranked in the top 4 of Stack OverFlows's most used programming languages (4th in 2022) and top 5 most loved programming languages by developers. Python is also synonymous with modern data science and is used extensively in research and development of computational methods and applications. We selected streamlit, over alternatives, for example, dash or shiny for Python, due to relatively shallow learning curves, and speed at which new users can build and deploy models. Given the code-based nature of our work we illustrate the implementation of streamlit in the form of a step-by-step tutorial.

We describe the steps involved in setting up a Python simulation model so that it is ready to incorporate into a web app, allowing user interactive simulation, basic tabular results presentation, basic interactive chart creation to display results, and use of our methods to create a large number of simulation experiments and run them in a single batch.

Our method is designed to be general to any DES package in Python. We primarily illustrate our method using a model built in simpy. We illustrate how our method supports interchangeability of models using a second simulation package called ciw and provide a full implementation in our *Extended data*[31].

Although our case study model is simple, the methods we describe are applicable across more complex simulations. To illustrate, our online *Extended data*[31] include an application to a second case study with time dependent arrivals, and multiple activities and classes of patient.

Case study: an urgent care call centre model

Our model is a stylised model of a typical urgent care call centre. Patient calls arrive at random to a call centre. Operators answer the calls, with a first in first out queue, triage the patient, interview following a standard script, and provide a call designation; for example, if the patient should travel to an emergency department, book an appointment in primary care with a General Practitioner (family doctor), or if a call back from a nurse is needed. If a caller needs to speak to a nurse they enter a first in first out queue until a nurse is available. Figure 1 illustrates patient flow in the model and use of resources.

Simulation software: simpy

Simpy is a DES package implemented in Python. It uses a process-based simulation worldview. DES models are built by defining Python generator functions and logic to request and return resources. Although simpy provides a full DES engine, the package does not contain many of the additional features offered by a full commercial simulation package, such as Arena or Simul8, including a user interface, instead simpy offers a lightweight, flexible tool that can be integrated with the rest of the Python data science ecosystem. It has been used to model a wide variety of health condition pathways and healthcare operations; for example, the coronavirus disease 2019 (COVID-19)[32], renal[12], stroke[33,34], heart failure[35], cancer care[36], end of life care[13], and operating theatre management[24,37].

Web app technology: streamlit

Streamlit is a software package to build web applications (apps) in Python. It has been designed to make web apps accessible to non-experts. A typical streamlit application is a short Python script.

Web apps built with streamlit can be interactive. Deployment of models to users could be as simple as a local Python script that provides a browser based front-end to a model, hosting the app in streamlit's free to use community cloud, or as complex as deployment to paid infrastructure such as Google Cloud. Our method is applicable across all of these deployment methods. For simplicity we detail the building of a simple browser-based front-end to a simulation model that is run on a laptop or desktop.

Analysis environment

All code was written in Python 3.9.16. The DES model was implemented in simpy 4.0.1 and ciw 2.3.7. We used streamlit 1.23.0 for web apps. Manipulation and analysis of simulation results was conducted using *pandas* 2.0.2[38] and *numpy* 1.25.0[39]. Charts were created using *matplotlib* 3.7.1[40] and *plotly* 5.15.0.

The computational analyses were run on Intel i9-9900K CPU with 64GB RAM running Pop! OS 20.04 Linux.

Reproducibility and availability of code

In addition to the code presented in this paper we provide a more detailed online tutorial in a Jupyter book (https://health-data-science-or.github.io/simpy-streamlit-tutorial/). The code and book are archived using

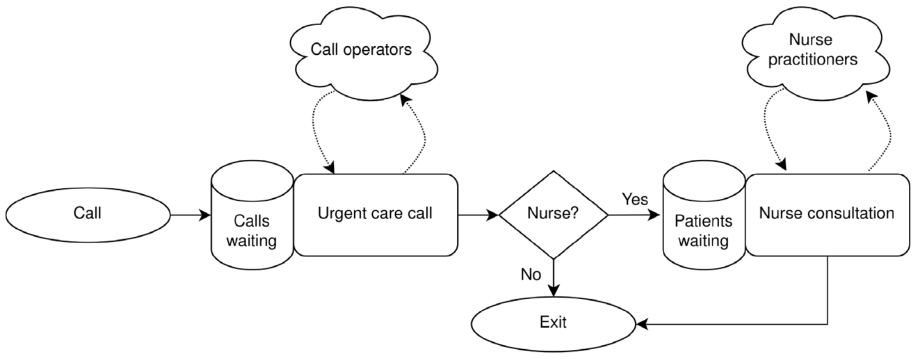


Figure 1. Urgent care call centre queuing model.

Zenodo[31]. Users can also obtain code from out GitHub repository: https://github.com/health-data-science-OR/simpy-streamlit-tutorial.

Overview of method

As others have stated, a web application should allow a user to change parameter values and rerun the model[22]. We add to this definition: a web app should provide a way for users to run more than one experiment at a time. By experiment we mean a change in one or more input parameters of the model. This type of experimentation is sometimes termed 'what-if' analysis and is highly relevant to operational research in healthcare[2]. Figure 2 illustrates the main concepts in the setup of our approach. We briefly outline these before describing implementation and code in our case study.

Encapsulating model logic in a Python module. Our method structures code and models so that a web app front-end does not depend on the simulation software package, or detailed logic of the model code. The opposite is also true, the implementation of the model is not affected by the logic or decisions made in the web app. Our approach ensures that a model is designed so that it can be run from a standard Python script, Jupyter notebook, or even a command line programme.

When combined with the model wrapper functions we propose in the next two sections, the simulation model becomes "swap-able". In the language of software development, this would be called *modular* code. This allows researchers to easily explore different implementations of a model in different software. For example, a model may have been developed in simpy, but is re-implemented in salabim or ciw without any change to the web app itself. To a lesser degree, this also simplifies the implementation of experiments that vary the process logic within a simulation as well as quantitative parameters; for example, modelling an emergency department with and without rapid assessment and treatment protocols as patients arrive. A caveat in this second case is that depending on the process logic chosen, the web app might require additional logic to vary the parameters that are available to users.

We propose two ways to encapsulate a full model in Python. The first is to create a standard Python module (*e.g.* called `model.py`) that contains the simulation model and associated functions. *I.e.* model logic, functions, classes, and constants (for example, functions to generate patient arrivals to a health care service). Second, a Python package could be created and possibly deployed via the Python Package Index. Once installed the package acts very much like a module with a user importing functionality to run model experiments. Of these two methods the simplest to maintain in a modelling study is a Python module.

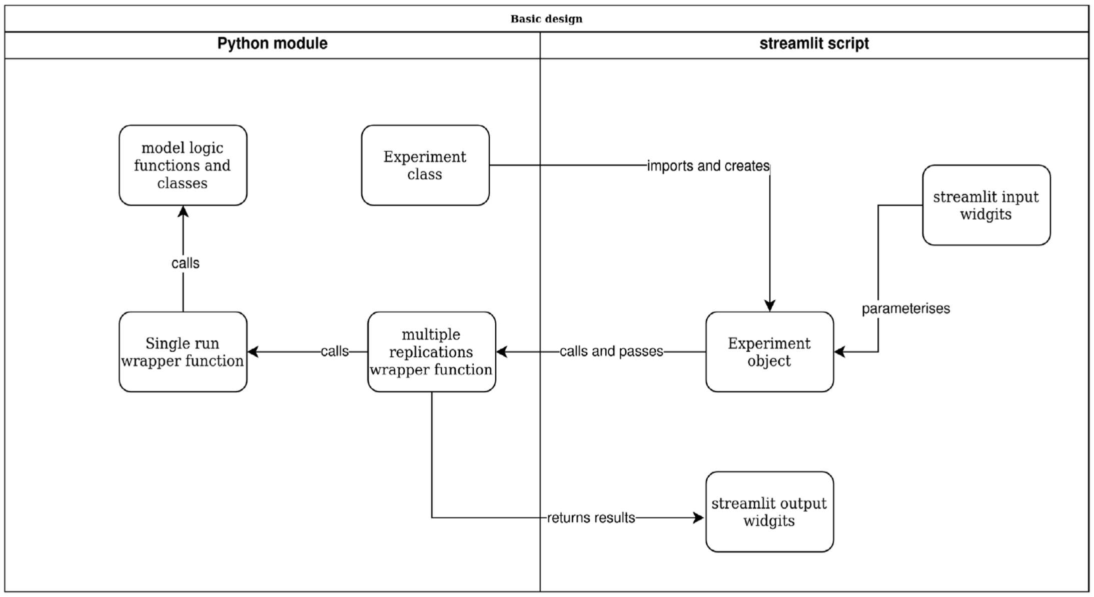


Figure 2. Model and web app structure overview.

Experiment class. In our approach, code that uses the simulation model does not directly import model logic. Instead it relies on an `Experiment` class and a wrapper function.

The `Experiment` class is tightly coupled to the design of the simulation model. It contains a configuration of the simulation model that can be executed to obtain results. Another way to describe an experiment is that it is a collection of input parameters for the simulation model. For example, in a study of a cancer pathway a user might set up two experiments: one containing the default number of screening slots on a given day and one containing an increment. The results of the experiments (such as patient waiting time and resource utilisation) can then be compared.

Wrapping up a run of the model with single run(). This function acts as a wrapper to run a simulated experiment and process end of run results. A key aspect of the design is that `single_run()` accepts an instance of `Experiment`. The code then sets up the model following the user-set parameters, executes a single run of the model and returns results. Listing 1 provides pseudo code for the design of this function. It is indicative of the general pattern you will follow; the exact details will vary depending on the simulation package in use.

```
SUBROUTINE single_run(experiment, run_length):
    setup_results_collection(experiment)
    model = create_simulation_model(experiment)
    results = run_model(model, run_length)
    summary_results = process_results(results)
    RETURN summary_results
END SUBROUTINE
```

Listing 1. Pseudo code for single run

multiple replications() wrapper function. The most likely usage scenario is that a model is run using multiple independent replications. This is a simple function that accepts an `Experiment` object and the number of replications to run. It then calls the `single_run()` function the appropriate number of times to create the distribution of results.

The streamlit script. The script contains all of the logic for the web app; for example, user settings, results tables and information about the simulation model.

The script imports the `Experiment` class, and the `multiple_replications()` wrapper function from the simulation model module. This is all that is needed to enable the interaction between the front-end and the model.

streamlit input widgets. The software has a number of standard input widgets available. For example, sliders, and text boxes to allow users to vary numeric values representing model parameters, and buttons to allow users to run models.

Experiment object. The script creates one or more instances of `Experiment`. The experiment object is parameterised by the script. For example, in an urgent care call centre two key parameters might be the number of call operators who handle incoming patient calls and nurses on duty and available for urgent call backs. The streamlit script might vary these inputs using the two values contained in streamlit input sliders.

Calling multiple replications(). The interface is set up so that a user clicks on a streamlit button and passes a parameterised instance of `Experiment` to `multiple_replications()` along with the number of replications to run.

streamlit output widgets. The `multiple_replications()` returns some form of results to the streamlit script; for example, a `pandas.Dataframe` containing results of each replication. The streamlit script then uses a output widget *e.g.* a `streamlit.dataframe` to display the results to the user.

Case study: model implementation

Getting simpy and streamlit

The code examples in this study have been created using a conda virtual environment. An analogy for a virtual environment is a box within an operating system. Inside the box, a specific version of Python is installed along with specific versions of data science and simulation libraries such as simpy. There can be multiple virtual environments within one machine. The approach attempts to minimise software dependency conflicts between projects. Full details of how to install our environment, along with a supporting video, is available in our *Extended data*[31]. We note if users are familiar with other virtual environment tools such as poetry then this can also be used.

Simulation model logic

To recreate the model from scratch, see our *Extended data*[31] for step by step instructions. Here we provide an overview of the main functions used to implement model logic.

The function `arrivals_generator()` is a simpy process for creating patient arrivals to the urgent care call centre. It contains a loop that continually samples the time until the next arrival using the Exponential distribution. Each patient arrival is allocated a new service process.

The function `service()` defines all of the logic after a caller has arrived. This includes requesting and queuing for a call operator resource, sampling call duration from a Triangular distribution, sampling if a nurse call back is required from a Bernoulli distribution, requesting and queuing for a nurse resource, and finally sampling nurse call duration from Uniform distribution. The function `service()` also collects patient waiting time and resource utilisation as the simulation executes.

Coding the Experiment class

The `Experiment` class encapsulates a set of input parameters that configure the simulation model. The overall design of a class representing an experiment will depend on the simulation study, and to some extent user preference. Here we advocate one key design principal:

- Make use of default values for input parameters, either from constant variables, or read in from file.

Lines 6–20 of Listing 2 illustrate the use of default parameters when creating an instance of `Experiment`; for example, `N_NURSES` provides a default number of nurses. In the implementation these are variables with module level scope. They are combined with Python's optional arguments for methods. This approach means that it is simple to create experiments with default or new parameter values. Listing 3 illustrates the method across three experiments: an experiment using all defaults; an experiment that varies the number of call operators; and an experiment that varies the number of call operators and nurses.

```
 1 class Experiment:
 2     """
 3     Parameter class for simulation model
 4     """
 5
 6     def __init__(
 7         self,
 8         n_operators=N_OPERATORS,
 9         n_nurses=N_NURSES,
10         mean_iat=MEAN_IAT,
11         call_low=CALL_LOW,
12         call_mode=CALL_MODE,
13         call_high=CALL_HIGH,
14         chance_callback=CHANCE_CALLBACK,
15         nurse_call_low=NURSE_CALL_LOW,
16         nurse_call_high=NURSE_CALL_HIGH,
17         arrival_seed=None,
18         call_seed=None,
19         callback_seed=None,
20         nurse_seed=None,
21     ):
```

```python
        """
        The init method sets up our defaults, resource
        counts, dists, + result collection objects.
        """
        # no. resources
        self.n_operators = n_operators
        self.n_nurses = n_nurses

        # create distribution objects
        self.arrival_dist = Exponential(mean_iat,
                                        arrival_seed)
        self.call_dist = Triangular(call_low, call_mode,
                                    call_high, call_seed)

        self.callback_dist = Bernoulli(chance_callback,
                                       callback_seed)

        self.nurse_dist = Uniform(nurse_call_low,
                                  nurse_call_high,
                                  nurse_seed)

        # resources
        # these variable are placeholders.
        self.operators = None
        self.nurses = None

        # initialise results to zero
        self.init_results_variables()

    def init_results_variables(self):
        """
        Initialise all of the experiment variables used
        in results collection. This method is called at
        the start of each replication.
        """
        # variable used to store results of experiment
        self.results = {}
        self.results["waiting_times"] = []

        # total operator usage time
        self.results["total_call_duration"] = 0.0

        # nurse sub process results collection
        self.results["nurse_waiting_times"] = []
        self.results["total_nurse_call_duration"] = 0.0
```

Listing 2. Experiment class

```python
default_experiment = Experiment()
extra_operator = Experiment(n_operators=14)
extra_operator_and_nurse = Experiment(n_operators=14, n_nurses=10)
```

Listing 3. Example usage of Experiment

Coding the wrapper functions

The function `single_run()` is detailed in Listing 4. The function accepts an instance of `Experiment` and optionally a user specified simulation run length. Logically the function is a very simple wrapper for a single replication of the the model configured using the input parameters in the model. It returns a summary of the results from that replication (*e.g.* the mean operator waiting time in the replication).

In our simpy example, lines 27 – 38 creates a simulation environment, operator and nurse resources, sets up `arrivals_generator()` as a simpy process, and executes the model. Lines 41–60 are executed when the simulation is complete and calculate summary variables. Note these might preferably be refactored into an `end_of_run_results()` function for instances with a large number of model metrics.

```
 1 def single_run(experiment,
 2                 rc_period=RESULTS_COLLECTION_PERIOD):
 3     """
 4     Perform a single run of the model and return
 5     a dictionary of results
 6 
 7     Parameters:
 8     -----------
 9     experiment: Experiment
10         The experiment/paramaters to use with model
11 
12     rc_period: float, optional
13                (default=RESULTS_COLLECTION_PERIOD)
14         Results collection period for simulation.
15 
16     Returns:
17     --------
18     dict
19     """
20     # results dictionary.  Each KPI is a new entry.
21     run_results = {}
22 
23     # reset all results variables to zero and empty
24     experiment.init_results_variables()
25 
26     # environment is (re) created inside single run
27     env = simpy.Environment()
28 
29     # create the resources
30     experiment.operators = simpy.Resource(env,
31                                           experiment.n_operators)
32 
33     experiment.nurses = simpy.Resource(env,
34                                        experiment.n_nurses)
35 
36     # setup arrivals_generators as simpy process
37     env.process(arrivals_generator(env, experiment))
38     env.run(until=rc_period)
39 
40     # end of run results: calculate mean waiting time
41     run_results["01_mean_waiting_time"] = np.mean(
42         experiment.results["waiting_times"]
43     )
44 
45     # end of run results: calculate mean operator utilisation
46     run_results["02_operator_util"] = (
47         experiment.results["total_call_duration"]
48         / (rc_period * experiment.n_operators)
49     ) * 100.0
50 
51     # end of run results: nurse waiting time
52     run_results["03 _mean_nurse_waiting_time"] = np.mean(
53         experiment.results["nurse_waiting_times"]
54     )
55 
```

```python
    # end of run results: calculate mean nurse utilisation
    run_results["04 _nurse_util"] = (
        experiment.results["total_nurse_call_duration"]
        / (rc_period * experiment.n_nurses)
    ) * 100.0

    # return the results from the run of the model
    return run_results
```

Listing 4. single run() wrapper

The `multiple_replications()` function is simpler than `single_run()`. It is detailed in Listing 5. In line 26 a Python list comprehension is used to repeatedly loop and execute `single_run()` and create a Python list. Each item in the list is a `dict` containing the results of a single replication of the model. Lines 30 to 32 then format the results as `pandas.Dataframe`.

```python
 1 def multiple_replications(
 2     experiment, rc_period=RESULTS_COLLECTION_PERIOD,
 3     n_reps=5):
 4     """
 5     Perform multiple replications of the model.
 6 
 7     Params:
 8     ------
 9     experiment: Experiment
10         The experiment/paramaters to use with model
11 
12     rc_period: float, optional
13         (default = DEFAULT_RESULTS_COLLECTION_PERIOD)
14         results collection period.
15         the number of minutes to run the model to collect results
16 
17     n_reps: int, optional (default=5)
18         Number of independent replications to run.
19 
20     Returns:
21     --------
22     pandas.DataFrame
23     """
24 
25     # loop over single run to generate lisy of results dicts.
26     results = [single_run(experiment, rc_period)
27         for rep in range(n_reps)]
28 
29     # format and return results in a dataframe
30     df_results = pd.DataFrame(results)
31     df_results.index = np.arange(1, len(df_results) + 1)
32     df_results.index.name = "rep"
33     return df_results
```

Listing 5. multiple replications() wrapper

Code to test the model

Before moving on to building a web app it is recommended that a basic Python script is created to test the code. Listing 6 assumes that the model code and wrapper functions have been stored in a Python module called `model.py`. Line 1 illustrates that all that a model user needs do is import `Experiment` and `multiple_replications()`. In our example, the user is only interested in varying the number of operators, nurses, and the chance of a callback. Lines 14–20 then parameterise the experiment and execute the model.

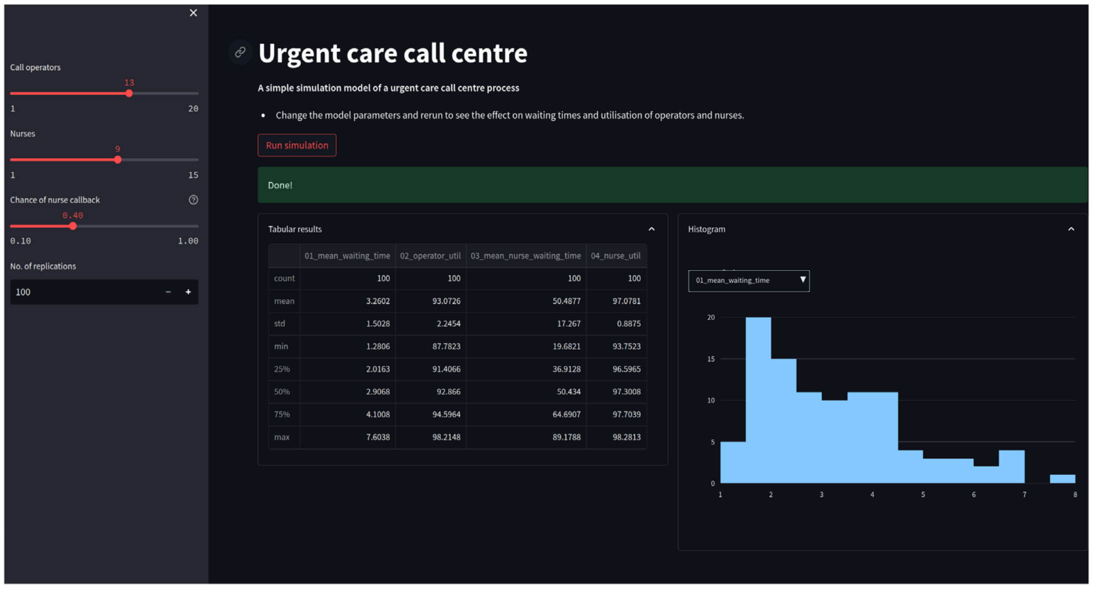
In a real simulation project it is recommended that more extensive code testing and verification of model logic is conducted. The full breadth of testing strategies available is out of scope for this tutorial. Interested readers are directed to the code testing section of The Turing Way[41] and to an introduction to Test Driven Development and Unit Testing in the context of computer simulation[42].

```python
from model import Experiment, multiple_replications

# set number of resources
n_operators = 13
n_nurses = 9

# set chance of nurse
chance_callback = 0.4

# set number of replications
n_reps = 5

# create experiment
exp = Experiment(
    n_operators=n_operators, n_nurses=n_nurses,
    chance_callback=chance_callback
)

# run multiple replications of experment
results = multiple_replications(exp, n_reps=n_reps)

# show results
print(results.describe())
```

Listing 6. A basic Python script

Converting to an interactive web app

An example web app, for the urgent care call centre, is illustrated in Figure 3. An interactive version of the app is also available via Streamlit's community cloud https://simpy-app-tutorial.streamlit.app.



Call operators: 13 (1–20)

Nurses: 9 (1–15)

Chance of nurse callback: 0.40 (0.10–1.00)

No. of replications: 100

Urgent care call centre

A simple simulation model of a urgent care call centre process

- Change the model parameters and rerun to see the effect on waiting times and utilisation of operators and nurses.

Done!

Tabular results

	01_mean_waiting_time	02_operator_util	03_mean_nurse_waiting_time	04_nurse_util
count	100	100	100	100
mean	3.2602	93.0726	50.4877	97.0781
std	1.5028	2.2454	17.267	0.8875
min	1.2806	87.7823	19.6821	93.7523
25%	2.0163	91.4066	36.9128	96.5965
50%	2.9068	92.866	50.434	97.3008
75%	4.1008	94.5964	64.6907	97.7039
max	7.6038	98.2148	89.1788	98.2813

Figure 3. Urgent call centre DES web app. Shown is a screen shot of app running in a browser window. The app includes sliders for varying input parameters, and a interactive histogram to display results.

In summary the web app provides the following functionality:

1. Sliders will allow a user to vary the number of operators, nurses, and the probability of a nurse callback;
2. For a given configuration, run an experiment of the model when a button is clicked;
3. The model will provide feedback to the user to let them know an experiment is running and when it has completed.
4. Display the summary statistics of the results of an experiment in a table and interactive histogram.

We also provide a title and some explanatory text for the model. For readers who wish to reproduce the web app, we recommend that Listing 6 is used as a starting point. Users can then incrementally makes changes to the script as we outline below. The script should be renamed `basic_app.py`.

Step 1: import streamlit and create a title

At the top of the script add `import streamlit as st`. This will mean the script can now be launched as a streamlit app. We will also include a title for the app using `st.title('Urgent care call centre')`. The app can now be launched from the command prompt (terminal on Linux and Mac): `streamlit run basic_app.py`

Step 2: Adding a button and displaying results

At the moment the model runs once - when the app starts. We will now add a `st.button` widget control when the app runs. In streamlit this is implemented using a Python if statement that checks the boolean value of the button. The use of a conditional may appear unintuitive at first, and is related to the way streamlit executes a script. We explain this in more detail in the next step.

After the model has executed the experiment, we need a way to display results to a user. We know that `multiple_replications()` returns a tabular result in the form of a dataframe. So we opt to use `st.dataframe` to display a formatted table to the user. This will appear underneath the button.

Listing 7 details the code from steps 1 and 2. The execution of the model is implemented in lines 31 to 36.

```
 1  # ###################################################################
 2  # MODIFICATION: import streamlit
 3  import streamlit as st
 4
 5  # ####################################################################
 6  from model import Experiment, multiple_replications
 7
 8  # ###################################################################
 9  # MODIFICATION: We add in a title for our web app's page
10  st.title("Urgent care call centre")
11  ####################################################################
12
13  # set number of resources
14  n_operators = 13
15  n_nurses = 9
16
17  # set chance of nurse
18  chance_callback = 0.4
19
20  # set number of replications
21  n_reps = 5
22
23  # create experiment
24  exp = Experiment(
25      n_operators=n_operators, n_nurses=n_nurses,
26      chance_callback=chance_callback
27  )
28
```

```
####################################################################
# MODIFICATION: press a streamlit button to run the model
if st.button("Run simulation"):
    # run multiple replications of experment
    results = multiple_replications(exp, n_reps=n_reps)

    # show results
    st.dataframe(results.describe())
####################################################################
```

Listing 7. basic app.py

Step 3: Making the experiment interactive

We will now modify our app so that it is interactive; *i.e.*, a user will be able to vary the parameters on screen to run custom experiments. We will do this using `streamlit.slider` and `st.number_input` widgets. Both of these functions display a streamlit input widget and returns a numeric value that can be assigned to a variable. Listing 8 illustrates the creation of a slider and number input. The variables `n_nurses` and `n_reps` are the same data type as seen in Listing 7. The functions take several mandatory and optional parameters. For example with a slider we have provided the text to display to a user, the minimum and maximum values, the default value and the step size between one value and the next.

```
n_nurses = st.slider('Nurses', 1, 15, 9, step=1)
n_reps = st.number_input("No. of replications", 100, 1_000, step=1)
```

Listing 8. Input widgets

It is now essential to understand how streamlit executes a script. Each time a user updates the value of a slider or clicks the run button streamlit executes the full Python script file *i.e.* from top to bottom. This means, for example, that the integer value of `n_nurses` changes each time the slider is moved. We can now also make sense of the use of an if statement with the `st.button` function. When the button is clicked it is assigned the value True. streamlit then executes the full script and will also execute the conditional logic contained under the if statement. When a slider is changed the button has a value of False so the model will not run unnecessarily. Listing 9 details the code within the modified script.

```
import streamlit as st
from model import Experiment, multiple_replications

# We add in a title for our web app's page
st.title("Urgent care call centre")

# ###################################################################
# MODIFICATION: user experiment via app

# set number of resources
n_operators = st.slider("Call operators", 1, 20, 13, step=1)
n_nurses = st.slider("Nurses", 1, 15, 9, step=1)

# set chance of nurse
chance_callback = st.slider(
    "Chance of nurse callback", 0.1, 1.0, 0.4, step=0.05)

# set number of replications
n_reps = st.number_input("No. of replications", 100, 1_000, step=1)

# ####################################################################

```

```python
# create experiment
exp = Experiment(
    n_operators=n_operators, n_nurses=n_nurses,
    chance_callback=chance_callback
)

# A user must press a streamlit button to run the model
if st.button("Run simulation"):
    # run multiple replications of experment
    results=multiple_replications(exp, n_reps=n_reps)

    # show results
    st.dataframe(results.describe())
```

Listing 9. interactive app.py

Step 4: Improving usability

To help usability, streamlit provides a number of simple features. We will implement three of these: a side bar that holds all of the input widgets, feedback to user that the model is running in the background, and a success box to report that the model run has completed.

A side bar is a section of the app that can be hidden or expanded depending on user preference, added using a Python with statement and `st.sidebar`. Its implementation can be seen in Listing 10 lines 10 to 21. Note that the widget code has been indented to ensure that it falls within the with statement block.

The second modification uses another with statement, this time combined with `st.spinner`. The spinner provides a simple animated prompt to users while the simulation is running. The code falls underneath the button logic for running the simulation model and can be seen in line 32 in Listing 10. Note again the indentation of the logic that will execute while the spinner is displayed on line 35. After the experiment is completed an st.success function is used to display a message of ‘Done’ to the user.

```python
 1 import streamlit as st
 2 from model import Experiment, multiple_replications
 3
 4 # We add in a title for our web app's page
 5 st.title("Urgent care call centre")
 6
 7 # ##################################################################
 8 # MODIFICATION: side bar
 9
10 with st.sidebar:
11     # set number of resources
12     n_operators = st.slider("Call operators", 1, 20, 13, step=1)
13     n_nurses = st.slider("Nurses", 1, 15, 9, step=1)
14
15     # set chance of nurse
16     chance_callback = st.slider(
17         "Chance of nurse callback", 0.1, 1.0, 0.4, step=0.05
18     )
19
20     # set number of replications
21     n_reps = st.slider("No. of replications", 5, 100, step=1)
22
23 # ##################################################################
24
25 # create experiment
26 exp = Experiment(
27     n_operators=n_operators, n_nurses=n_nurses,
28     chance_callback=chance_callback
29 )
30
```

```
# A user must press a streamlit button to run the model
if st.button("Run simulation"):
    # ############################################################
    # MODIFICATION: add a spinner and then display success box
    with st.spinner("Simulating the urgent care system..."):
        # run multiple replications of experment
        results = multiple_replications(exp, n_reps=n_reps)

    st.success("Done !")
    # ############################################################

    # show results
    st.dataframe(results.describe())
```

Listing 10. tidy app.py

Step 5: Interactive visualisation of results

There are many options for visualisation in Python: *e.g.* matplotlib, plotly, altair, or bokeh. These are not specific to streamlit, and a full introduction to these packages is out of scope. Here we provide a simple example chart using plotly: a free and open source library for interactive charting. A matplotlib chart would be a fine choice as an alternative; however, we opt for an interactive chart to maximise the benefits offered from a web app. Our supplementary material details the coding of a function called `create_user_filtered_hist`. The chart is illustrated on the right hand side of Figure 3. The function accepts the `results` dataframe returns from the experiment. The histogram chart displays the distribution of a single performance measure across replications. The user is able to select a performance measure displayed from the full list using a drop down list. Note this is all achieved through plotly and avoids any complexity issues with the persistence of simulation results in the web app.

Listing 11 provides an extract of a streamlit script that makes use of the *plotly* function (see *Extended data* for the full listing). In the script we provide two further approaches to clean up the user interface and improve usability. We create two columns for side by side result display using `st.columns(2)` (line 8). We place the results table and plotly chart in these columns.

```
 1 if st.button("Run simulation"):
 2
 3     with st.spinner('Simulating the urgent care system... '):
 4         results = multiple_replications(exp, n_reps=n_reps)
 5     st.success('Done !')
 6
 7     # create column layout
 8     col1, col2 = st.columns(2)
 9     with col1.expander('Tabular results', expanded=True):
10         # show tabular results
11         st.dataframe(results.describe())
12
13     with col2.expander('Histogram', expanded=True):
14         # create histogram and call plotly function
15         fig = create_user_filtered_hist(results)
16         st.plotly_chart(fig, use_container_width=True)
```

Listing 11. Modified results presentation code

Additional functionality

Loading and running multiple experiments

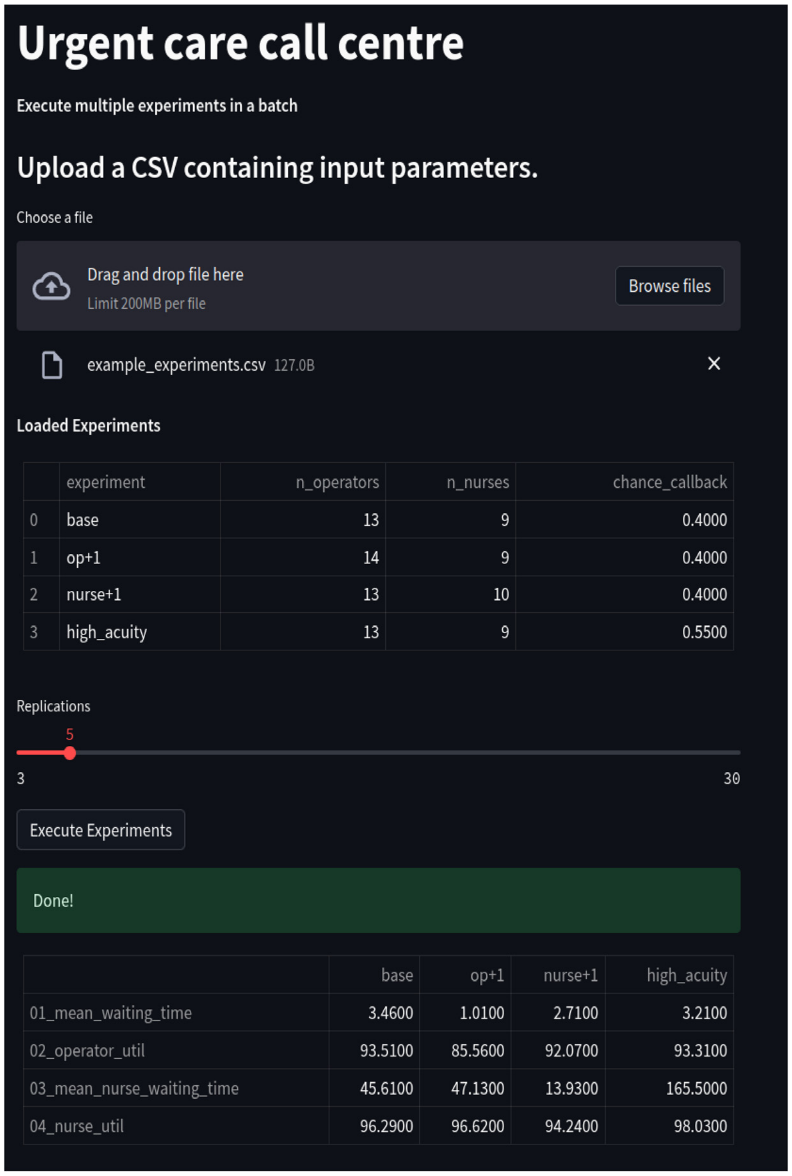
The `Experiment` class provides a simple way to run multiple experiments in a batch. To do so we can create multiple instances of `Experiment`, each with a different set of inputs for the model. These are then executed in a loop.

A method to implement this in streamlit is to upload a Comma Separated Value (.CSV) file containing a list of experiments to the web app. This can be stored internally as a `pandas.Dataframe` and displayed using a streamlit widget such as `st.table`. A user can then edit experiment files locally on their own machine (for example, using a spreadsheet software, or using CSV viewer/editor extensions for Jupyter-Lab or Visual Studio Code) and upload, inspect, and run, and view results in the app. Example code to implement the approach is included in the *Extended data*[31]. Figure 4 illustrates the web app it creates.

Replacing simpy with ciw

Our DES model has been implemented in simpy, but our approach allows for this to be replaced if required. A high-quality alternative to simpy is ciw[28]: a FOSS package for discrete-event simulation of queuing networks. A full overview of ciw is out of scope; we refer interested readers to its extensive online documentation.

To make the change we only need to update `model.py`. The streamlit script does not need to be modified. The main modifications are to the `single_run()` function -that now creates an instance of a ciw network


Urgent care call centre
Execute multiple experiments in a batch
Upload a CSV containing input parameters.
Choose a file
Drag and drop file here
Limit 200MB per file
Browse files
example_experiments.csv 127.0B
Loaded Experiments

	experiment	n_operators	n_nurses	chance_callback
0	base	13	9	0.4000
1	op+1	14	9	0.4000
2	nurse+1	13	10	0.4000
3	high_acuity	13	9	0.5500

	base	op+1	nurse+1	high_acuity
01_mean_waiting_time	3.4600	1.0100	2.7100	3.2100
02_operator_util	93.5100	85.5600	92.0700	93.3100
03_mean_nurse_waiting_time	45.6100	47.1300	13.9300	165.5000
04_nurse_util	96.2900	96.6200	94.2400	98.0300

Figure 4. Multiple experiments app for the Urgent Care Call Centre. Shown is a screen shot of the app running in a browser window. The app has loaded and executed a CSV file containing four experiments. A table summarises experiment results.

model and runs it for one replication - and `Experiment` that now uses ciw's built in distributions. See the the online *Extended data*[31] for full code listings. The ciw implementation provides qualitatively identical results and the same conclusions as simpy.

Discussion

We demonstrate how to build a user-friendly interface to DES models coded in Python using streamlit. Our work aims to enhance Open Science within health service delivery and OR. Our approach supports researchers to share their computer models with other research teams and the NHS. Our Python method work is complementary to existing studies in Health Economic Evaluation and Markov modelling. Although no DES examples are available, modellers aiming to use an R DES package such as Simmer may wish to consult these sources for support[22,23].

Strengths of the approach

Strengths of our approach include a simple way to structure DES model Python code to allow for reuse, extension, and replacement, of models without the need to heavily modify a user interface. We demonstrated the latter by switching from simpy to ciw, although the approach would work just as well with different DES packages (*e.g.* salabim). Our method focus is DES, but the wrapper functions and experiment structure may generalise to other methods that use simulation. A closely related candidate method is Agent Based Simulation within healthcare as it makes use of stochastic sampling, multiple replications and experiments.

The apps created in the previous sections can be shared with other researchers and the NHS using a very simple method: code and virtual environment files are committed to GitHub (or alternative such as GitLab). These can then be downloaded and run on a laptop or desktop. Python has an advantage that it is a cross-platform language, *i.e.* code will run on Microsoft Windows, Mac OS or Linux[10]. Alternatively users can make use of streamlit community cloud, containerisation (*e.g* using Docker); or creating an .exe file (on Microsoft Windows). There is also the more complex option to host the web app remotely and pay for more powerful computing infrastructure (*e.g.* Google Cloud).

Another strength of our approach is that our `multiple_replications` function is easily parallelisable using Python's *joblib* library. If the model is used locally, there may be a large time saving benefit of taking this approach, *i.e.* running parallel replications on a laptop with 20 virtual cores. However, in general these benefits are unlikely to transfer to web based models[10] without using a paid tier offered by a web app hosting provider. For example, Streamlit's community cloud or ShinyApps.io free tier deployment will limit model execution to a single CPU.

Limitations

Our tutorial has a number of limitations. One issue we do not consider are projects where data, used to derive parameters or the parameters themselves used by the simulation model, may need to be kept private due to governance or ethical reasons. In these instances hosting a model interface on a service such as streamlit community cloud is more challenging. This is a contemporary problem for the UK NHS. Initiatives such as OpenSafely have emerged to tackle reproducible pipelines and could support deriving model parameters[43]. There is likely much that can be learnt from the Health Economic Evaluation community and research in R where models access secure data and parameters remotely using a secure API[44].

While we have discussed creating model interfaces and wrapper functions for Python DES models, we have not been prescriptive about how these models are logically structured internally or what else should be included. Others have argued that standardisation of code and its organisation will benefit transparency and reuse in Health Economic Evaluation[45,46]. This includes naming conventions, unit testing, and project documentation. The inclusion of comprehensive documentation has been identified as a primary motivator of reuse of code[47].

Our tutorial focused on the mechanics of building a usable *streamlit* interface for DES models, but we have not described detailed customisation options for interfaces. These might, for instance, be important for models where NHS or research funder branding are needed. Basic options provided by *streamlit* include the ability to display NHS or funder logos using the `st.image()` function. More flexible and detailed control of appearance can be accessed via *streamlit*'s theming options, *e.g.*, controlling font, and background colour of different sections or pages.

When compared to commercial DES software packages that are commonly used in health research, such as Simul8, or AnyLogic, a limitation of our approach is that we do not display a dynamic patient pathway or queuing network that updates as the model runs a single replication. This is termed Visual Interactive Simulation (VIS) and can help users understand where process problems and delays occur in a patient pathway[48]; albeit with the caveat that single replications can be outliers. A potential FOSS solution compatible with a browser-based app could use a Python package that can represent a queuing network, such as NetworkX[49], and displaying results via matplotlib. If sophisticated VIS is essential for a FOSS model then researchers may need to look outside of web apps; for example, salabim provides a powerful FOSS solution for custom animation of DES models.

Conclusions and future work

One aim of Open Science is to increase the accessibility of research. The code for healthcare DES models, often tackling important treatment pathways, can be shared alongside publications, but requires specialist skills to use and run. Building on work from other health data science disciplines, we propose that web apps offer a user-friendly interface for healthcare DES models that increases research accessibility to the NHS and researchers from other disciplines. We provide an approach to structuring DES model code in Python and a worked example linking this to a streamlit interface. Further work could consider interface design, and visual interactive simulation.

Data availability

A detailed online tutorial with executable code is available online (https://health-data-science-or.github.io/simpy-streamlit-tutorial/).

Source code available from: https://github.com/health-data-science-OR/simpy-streamlit-tutorial

Archived source code at time of publication: https://doi.org/10.5281/zenodo.8193001[31]

License: MIT License

Acknowledgements

This report is independent research supported by the National Institute for Health Research Applied Research Collaboration South West Peninsula. The views expressed in this publication are those of the author(s) and not necessarily those of the National Institute for Health Research or the Department of Health and Social Care. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

1. Robinson S, Nance RE, Paul RJ, *et al.*: **Simulation model reuse: definitions, benefits and obstacles.** *Simul Model Pract Theory.* 2004; **12**(7–8): 479–94.
Publisher Full Text
2. Pitt M, Monks T, Crowe S, *et al.*: **Systems modelling and simulation in health service design, delivery and decision making.** *BMJ Qual Saf.* 2016; **25**(1): 38–45.
PubMed Abstract | **Publisher Full Text**
3. Salleh S, Thokala P, Brennan A, *et al.*: **Simulation Modelling in Healthcare: An Umbrella Review of Systematic Literature Reviews.** *Pharmacoeconomics.* 2017; **35**(9): 937–949.
PubMed Abstract | **Publisher Full Text**
4. Vázquez-Serrano JI, Peimbert-García RE, Cárdenas-Barrón LE: **Discrete-Event Simulation Modeling in Healthcare: A Comprehensive Review.** *Int J Environ Res Public Health.* 2021; **18**(22): 12262.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**
5. Forbus JJ, Berleant D: **Discrete-Event Simulation in Healthcare Settings: A Review.** *Modelling.* 2022; **3**(4): 417–33.
Publisher Full Text
6. Roy S, Venkatesan SP, Goh M: **Healthcare services: A systematic review of patient-centric logistics issues using simulation.** *J Oper Res Soc.* 2021; **72**(10): 2342–64.
Publisher Full Text
7. Monks T, Currie CSM, Onggo BS, *et al.*: **Strengthening the reporting of empirical simulation studies: Introducing the STRESS guidelines.** *J Simulation.* 2019; **13**(1): 55–67.
Publisher Full Text
8. Monks T, Harper A: **Computer model and code sharing practices in health-care discrete-event simulation: a systematic scoping review**. *J Simulation.* 2023.
Publisher Full Text
9. Pouwels XGLV, Sampson CJ, Arnold RJG, *et al.*: **Opportunities and Barriers to the Development and Use of Open Source Health Economic Models: A Survey.** *Value Health.* 2022; **25**(4): 473–479.
PubMed Abstract | **Publisher Full Text**
10. Byrne J, Heavey C, Byrne PJ: **A review of Web-based simulation and supporting tools.** *Simul Model Pract Theory.* 2010; **18**(3): 253–76.
Publisher Full Text
11. Dagkakis G, Heavey C: **A review of open source discrete event simulation software for operations research.** *J Simulation.* 2016; **10**(3): 193–206.
Publisher Full Text

12. Allen M, Bhanji A, Willemsen J, *et al.*: **A simulation modelling toolkit for organising outpatient dialysis services during the COVID-19 pandemic.** *PLoS One.* 2020; **15**(8): e0237628.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

13. Chalk D, Robbins S, Kandasamy R, *et al.*: **Modelling palliative and end-of-life resource requirements during COVID-19: implications for quality care.** *BMJ Open.* 2021; **11**(5): e043795.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

14. Anagnostou A, Groen D, Taylor SJE, *et al.*: **FACS-CHARM: A Hybrid Agent-Based and Discrete-Event Simulation Approach for Covid-19 Management at Regional Level**. In: *2022 Winter Simulation Conference (WSC)*. IEEE, 2022; 1223–34.
Publisher Full Text

15. Wood RM, Moss SJ, Murch BJ, *et al.*: **Optimising acute stroke pathways through flexible use of bed capacity: a computer modelling study.** *BMC Health Serv Res.* 2022; **22**(1): 1068.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

16. Mohd S, Mustafee N, Madan K, *et al.*: **Leveraging Multi-tier Healthcare Facility Network Simulations for Capacity Planning in a Pandemic**. 2021.
Publisher Full Text

17. Crowe S, Grieco L, Monks T, *et al.*: **Here's something we prepared earlier: Development, use and reuse of a configurable, inter-disciplinary approach for tackling overcrowding in NHS hospitals.** *J Oper Res Soc.* 2023; 1–16.
Publisher Full Text

18. Harvard S, Adibi A, Easterbrook A, *et al.*: **Developing an Online Infrastructure to Enhance Model Ac- cessibility and Validation: The Peer Models Network.** *Pharmacoeconomics.* 2022; **40**(10): 1005–1009.
PubMed Abstract | **Publisher Full Text**

19. Janssen MA, Alessa LN, Barton M, *et al.*: **Towards a Community Framework for Agent-Based Modelling.** *J Artif Soc Soc Simul.* 2008; **11**(2): 6.
Reference Source

20. Knight V, Palmer G: **Applied Mathematics with Open-Source Software**. Chapman & Hall/CRC Series in Operations Research; 2022.
Publisher Full Text

21. Harper A, Monks T: **A Framework to Share Healthcare Simulations on the Web Using Free and Open Source Tools and Python**. In: *Proceedings of SW21 The OR Society Simulation Workshop*. Operational Research Society, 2023; 250–60.
Reference Source

22. Smith R, Schneider P: **Making health economic models Shiny: A tutorial [version 2; peer review: 2 approved].** *Wellcome Open Res.* 2020; **5**: 69.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

23. Wojciechowski J, Hopkins AM, Upton RN: **Interactive pharmacometric applications using R and the shiny package.** *CPT Pharmacometrics Syst Pharmacol.* 2015; **4**(3): e00021.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

24. Harper A, Monks T, Wilson R, *et al.*: **Post-Covid Orthopaedic Elective Resource Planning using Simulation Modelling.** *medRxiv.* 2023; 2023–05.
Publisher Full Text

25. Tyler JMB, Murch BJ, Vasilakis C, *et al.*: **Improving uptake of simulation in healthcare: User-driven development of an open-source tool for modelling patient flow.** *J Simul.* 2023; **17**(6): 765–782.
Publisher Full Text

26. Team SimPy: **SimPy 3.0.11**. 2020.
Reference Source

27. van der Ham R: **salabim: discrete event simulation and animation in Python.** *J Open Source Softw.* 2018; **3**(27): 767.
Publisher Full Text

28. Palmer GI, Knight VA, Harper PR, *et al.*: **An open-source discrete event simulation library.** *J Simul.* 2019; **13**(1): 68–82.
Publisher Full Text

29. NHS England: **NHS England Open Source Programme**. 2022.
Reference Source

30. Goldacre B, Morely J: **Better, broader, safer: using health data for research and analysis**. A review commissioned by the Secretary of State for Health and Social Care. Department of Health and Social Care, 2022.
Reference Source

31. Monks T, Harper A: **SimPy and StreamLit Tutorial Materials for Health-care Discrete-Event Simulation.** *Zenodo.* [Data]; 2023.
http://www.doi.org/10.5281/zenodo.8193001

32. Bovim TR, Gullhav AN, Andersson H, *et al.*: **Simulating emergency patient flow during the COVID-19 pandemic.** *J Simul.* 2023; **17**(4): 407–21.
Publisher Full Text

33. Ren Y, Phan M, Luong P, *et al.*: **Application of a computational model in simulating an endovascular clot retrieval service system within regional Australia.** *J Med Imaging Radiat Oncol.* 2021; **65**(7): 850–7.
PubMed Abstract | **Publisher Full Text**

34. Ren Y, Phan M, Luong P, *et al.*: **Geographic Service Delivery for Endovascular Clot Retrieval: Using Discrete Event Simulation to Optimize Resources.** *World Neurosurg.* 2020; **141**: e400–13.
PubMed Abstract | **Publisher Full Text**

35. Wise AF, Morgan LE, Heib A, *et al.*: **Modeling Of Waiting Lists For Chronic Heart Failure In The Wake Of The COVID-19 Pandemic.** In: *2021 Winter Simulation Conference (WSC)*. 2021; 1–11.
Publisher Full Text

36. Richardson DB, Cohn AEM: **Modeling the Impact of Make-ahead Chemotherapy Drug Policies through Discrete-Event Simulation.** In: *2018 Winter Simulation Conference (WSC)*. 2018; 2690–700.
Publisher Full Text

37. Hassanzadeh H, Boyle J, Khanna S, *et al.*: **A discrete event simulation for improving operating theatre efficiency.** *Int J Health Plann Manage.* 2023; **38**(2): 360–79.
PubMed Abstract | **Publisher Full Text**

38. McKinney W: **pandas: a foundational Python library for data analysis and statistics.** *Python for High Performance and Scientific Computing.* 2011; **14**.
Reference Source

39. van der Walt S, Colbert SC, Varoquaux G: **The NumPy Array: A Structure for Efficient Numerical Computation.** *Comput Sci Eng.* 2011; **13**(2): 22–30.
Publisher Full Text

40. Hunter JD: **Matplotlib: A 2D graphics environment.** *Comput Sci Eng.* 2007; **9**(3): 90–5.
Publisher Full Text

41. The Turing Way Community: **The Turing Way: A handbook for reproducible, ethical and collaborative research.** *Zenodo.* 2022; Accessed: 2023-10-18.
Publisher Full Text

42. Onggo BS, Karatas M: **Test-driven simulation modelling: A case study using agent-based maritime search-operation simulation.** *Eur J Oper Res.* 2016; **254**(2): 517–31.
Publisher Full Text

43. Williamson EJ, Walker AJ, Bhaskaran K, *et al.*: **Factors associated with COVID-19-related death using Open-SAFELY.** *Nature.* 2020; **584**(7821): 430–6.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

44. Smith R, Schneider P, Mohammed W: **Living HTA: Automating Health Economic Evaluation with R [version 2; peer review: 2 approved].** *Wellcome Open Res.* 2022; **7**: 194.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

45. Alarid-Escudero F, Krijkamp EM, Pechlivanoglou P, *et al.*: **A Need for Change! A Coding Framework for Improving Transparency in Decision Modeling.** *Pharmacoeconomics.* 2019; **37**(11): 1329–39.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

46. Smith R, Mohammed W, Schneider P: **Packaging cost-effectiveness models in R: A tutorial [version 1; peer review: 2 approved with reservations].** *Wellcome Open Res.* 2023; **8**: 419.
Publisher Full Text

47. den Eynden VV, Knight G, Vlad A, *et al.*: **Survey of Wellcome researchers and their attitudes to open research**. London: Wellcome Trust; 2016.
Reference Source

48. Bell PC, O'Keefe RM: **Visual interactive simulation: A methodological perspective.** *Ann Oper Res.* 1994; **53**(1): 321–42.
Publisher Full Text

49. Hagberg AA, Schult DA, Swart PJ: **Exploring Network Structure, Dynamics, and Function using NetworkX**. In: Varoquaux G, Vaught T, Millman J, editors. *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA, 2008; 11–15.
Reference Source

Open Peer Review

Current Peer Review Status: ✓ ✓ ✓

Version 2

Reviewer Report 28 December 2023

https://doi.org/10.3310/nihropenres.14669.r30915

© **2023 Smith R.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✓ **Robert Smith**
The University of Sheffield, Sheffield, England, UK

The authors have addressed all of the points I raised. In particular I think having the app available is really useful to give people context. I hope that this paper is useful in pushing the OR community towards open-source programming languages (even if it is R rather than Python!).

I have a couple of other comments, which need no action:

1) The authors stated that: "However, in general these benefits are unlikely to transfer to web based models[10] without using a paid tier offered by a web app hosting provider." However, this is not the case for R and shiny which I believe allows for analysis to be parallelized to multiple cores (although limited to a smaller number - I think maybe 4 or 8) on the free version of shinyapps.io.

2) The authors state that "There is likely much that can be learnt from the Health Economic Evaluation community and research in R where models access secure data and parameters remotely using a secure API[44]." This statement is true, but this functionality is not particularly commonplace, still being in early stages.

3) "Further work could consider interface design, and visual interactive simulation." Couldn't agree more, there is a recognition in the health economics industry that more research on how to articulate models (in terms of UX design and visualizations) is needed.

Thankyou again, I enjoyed reading this.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Application of methods from data-science to health economic evaluation.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 28 December 2023

https://doi.org/10.3310/nihropenres.14669.r30913

© 2023 Kritikos A. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✓

Apostolos Kritikos
Aristotle University of Thessaloniki, Thessaloniki, Greece

I want to congratulate the authors for their revision. I have no further comments.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: I would like to clarify, for full transparency, that, coming from the Open Source Software Engineering discipline, my reviews focus on the proposed idea of using Open Source Software solutions to boost digital models, enable collaboration, and promote open science.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Reviewer Report 17 October 2023

https://doi.org/10.3310/nihropenres.14611.r30520

© 2023 Kritikos A. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

?

Apostolos Kritikos
Aristotle University of Thessaloniki, Thessaloniki, Greece

The authors of this manuscript are conducting research on a very interesting challenge, that is, how to utilize open source software to provide an easy and user-friendly way for the health science community to work with models, share their findings and collaborate with other researchers or NHS decision makers.

This is a well written manuscript and I very much enjoyed reading it and gained knowledge while doing so. I would like to clarify, for full transparency, that, coming from the Open Source Software Engineering discipline, my review will be focusing on the proposed idea of using open source software solutions to boost digital models, enable collaboration and promote open science.

Pros

- The manuscript is well written and structured.
- The main point of the research is clear.
- The authors are successfully justifying their software choices and provide a detailed explanation on how their solution works and can be reproduced.
- The references are carefully selected and up to date.
- The assets (diagrams, code, examples) are easy to follow and made available as open data / open source code (i.e Zenodo, GitHub).

Improvements

- Reading the manuscript, the extensibility of the solution is evident when it comes to multiplicity of experiments and UI capabilities (within the limitations of the streamlit environment, of course). It is suggested that the authors clarify more on whether this solution can expand outside the DES models domain.
- Furthermore, could the proposed solution utilize another programming language (i.e. R). This is not being made very clear in the manuscript.

Minor improvements

- The "Plain Language summary" section is appearing twice in the manuscript. I apologize to the authors beforehand, if this is forced by the template of the publication.
- Discussion section also includes some limitations of the work. I would propose that the authors split "Discussions" section to "Discussion" and "Limitations or Threats to validity". I would like to further propose that the limitations section is extended a bit with other limitations (potentially linked with our suggestions on the Improvements section of this review - if deemed necessary by the authors).
- Finally, it is suggested that the last section, titled "Conclusions" should be renamed to "Conclusions and Future work" for more clear context.

Is the rationale for developing the new method (or application) clearly explained?

Yes

Is the description of the method technically sound?

Yes

Are sufficient details provided to allow replication of the method development and its use by others?

Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: I would like to clarify, for full transparency, that, coming from the Open Source Software Engineering discipline, my review will be focusing on the proposed idea of using open source software solutions to boost digital models, enable collaboration and promote open science.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 15 Dec 2023
Thomas Monks

Response to Reviewer 3 (R3)

R3: This is a well written manuscript and I very much enjoyed reading it and gained knowledge while doing so.

We are very grateful for a reviewer with your open source software experience to provide feedback on our work. Many thanks for your time.

R3: Reading the manuscript, the extensibility of the solution is evident when it comes to multiplicity of experiments and UI capabilities (within the limitations of the streamlit environment, of course). It is suggested that the authors clarify more on whether this solution can expand outside the DES models domain.

Yes, we do believe this is the case for health service delivery research where a variety of research methods are employed for modelling and prediction. Our primary experience with web apps is using computer simulation for modelling of health services, but we believe that it is equally application in other areas we have worked such as forecasting. That said, we think we should be cautious of making claims here without evidence (tests/applications). We thought a compromise was to focus on agent based simulation as has several characteristics similar to DES. We have added the following text in the section on strengths:

- "Our method focus is DES, but the wrapper functions and experiment structure may generalise to other methods that use simulation. A closely related candidate method is Agent Based Simulation within healthcare as it makes use of stochastic sampling, multiple replications and experiments"

R3: Furthermore, could the proposed solution utilize another programming language (i.e. R). This is not being made very clear in the manuscript.

Thank you for raising this important point. Yes we are confident the approach would generalise to R, although the technology and packages used would be different. A study that greatly influenced our own is Smith and Schneider (2020) that is set in a Health Economic Evaluation and earlier in pharmacometrics Wojciechowski et al, (2015) contexts using Markov models. We have signposted readers to this work already (in the section on aims), but we have added the additional text in summary section of the discussion.

- "Our Python method work is complementary to existing studies in Health Economic Evaluation and Markov modelling. Although no DES examples are available, modellers aiming to use an R DES package such as Simmer may wish to consult these sources for support (Smith and Schneider 2020; Wojciechowski et al, 2015)

R3: The "Plain Language summary" section is appearing twice in the manuscript. I apologize to the authors beforehand, if this is forced by the template of the publication.

Thank you for identifying this problem with our manuscript. We think we made a mistake with the article template. We believe this is now fixed and the PDF should only show one plain English summary.

R3: Discussion section also includes some limitations of the work. I would propose that the authors split "Discussions" section to "Discussion" and "Limitations or Threats to validity". I would like to further propose that the limitations section is extended a bit with other limitations (potentially linked with our suggestions on the Improvements section of this review - if deemed necessary by the authors).

We agree this improves the manuscript and will help readers understand the caveats of our work. We have created a separate "limitations" section greatly expanded the discussion of limitations. Please see our responses to other reviewers for details of these additions.

R3: Finally, it is suggested that the last section, titled "Conclusions" should be renamed to "Conclusions and Future work" for more clear context.

As requested we have updated the title of the final section to "Conclusions and Future Work"

References:

- Smith R, Schneider P. Making health economic models Shiny: A tutorial.Wellcome Open Research. 2020;5(69)
- Wojciechowski J, Hopkins AM, Upton RN. Interactive pharmacometric applications using R and the shiny package. CPT: pharmacometrics & systems pharmacology. 2015;4(3):146-59.

Competing Interests: None

Reviewer Report 12 October 2023

https://doi.org/10.3310/nihropenres.14611.r30528

© **2023 Smith R.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Robert Smith
The University of Sheffield, Sheffield, England, UK

This is a very useful paper and the accompanying materials are very thorough. As someone who has advocated for web-based UIs for models in health economics its great to see the same methods applied in other areas (here OR). I hope this paper gets as much engagement and enthusiasm as we have benefitted from in health economics.

I tend to code in R more than Python, so my comments are more high level.

1. It would be useful to have deployed the app so that those that are not familiar with these web-based applications can get an idea of the output from the code (we did this in Smith & Schneider, 2020 and I think it did help with engagement to have people play & share the app they'd built online). This is described in the discussion as "There is also the more complex option to host the web app remotely and pay for more powerful computing infrastructure (e.g. Google Cloud)" but this complexity is all on the developer side, and it makes it much easier for the user (who may not understand Python/GitHub and may not be able or permitted to install onto their machine). The paper also mentions 'Web-apps' a large number of times and then therefore its surprising there wasn't more discussion of how to achieve this.

2. Related to the above, there could be much more discussion about the best way to share the interactive model with potential users. Some options are listed, but there are trade-offs involved, especially in relation to data-sharing, something that we tried to solve in (Smith, Schneider & Mohammed., 2023) https://wellcomeopenresearch.org/articles/7-194 using APIs to send models to data, rather than data being sent to models. I don't think we've really cracked it, or explained it particularly well to those who are not familiar with such methods. Therefore we are still exploring different solutions, as this is probably the biggest barrier to modelling transparency in our industry (and I suspect it is the same in healthcare OR). I would be keen to hear the authors' opinions & experiences in Healthcare Operational Research settings and potential solutions.

3. There is no discussion around documentation and testing of code. While I appreciate this is not the primary focus of the paper, if the authors are advocating for sharing the code via GitHub and users deploying the apps themselves (e.g. locally in on organization servers)

then model documentation & testing will be crucial (this is perhaps less crucial when the original authors deploy the app themselves - although remains best practice). We have just written a paper on building R packages for health economic models (Smith, Mohammed & Schneider, 2023) https://wellcomeopenresearch.org/articles/8-419 with the aim being that each model should be documented and tested by the original authors in a systematic way before being shared with external parties, I assume there is a similar expectation in OR and with models developed in Python?

4. "streamlit’s community cloud deployment will limit model execution to a single CPU." presumably this is just an issue with streamlit rather than a fundamental problem with web-based models. For example models deployed on AWS/Google Cloud can be run on multiple CPUs.

5. There is not much discussion around the customizability of the UI. Again, I accept this is not the primary focus of the paper but my experience is that organizations will require the user-interface to look and feel a certain way (often to match their brand) and therefore the feasibility of this using Streamlit may limit its uptake.

References
1. Smith R, Mohammed W, Schneider P: Packaging cost-effectiveness models in R: A tutorial. *Wellcome Open Research*. 2023; **8**. Publisher Full Text
2. Smith R, Schneider P, Mohammed W: Living HTA: Automating Health Technology Assessment with R. *Wellcome Open Research*. 2022; **7**. Publisher Full Text
3. Smith R, Schneider P: Making health economic models Shiny: A tutorial.*Wellcome Open Res*. 2020; **5**: 69 PubMed Abstract | Publisher Full Text

Is the rationale for developing the new method (or application) clearly explained?
Yes

Is the description of the method technically sound?
Yes

Are sufficient details provided to allow replication of the method development and its use by others?
Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Application of methods from data-science to health economic evaluation.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 15 Dec 2023

Thomas Monks

Response to Reviewer 2 (R2)

R2: This is a very useful paper and the accompanying materials are very thorough. As someone who has advocated for web-based UIs for models in health economics its great to see the same methods applied in other areas (here OR). I hope this paper gets as much engagement and enthusiasm as we have benefitted from in health economics

Your work was very influential on our own and we are very grateful to receive feedback from someone who has already worked in this area. Many thanks for your time to review the manuscript and for links to your recent work. Your work on R packages is very similar to our own current thinking and early work in Python.

R2: It would be useful to have deployed the app so that those that are not familiar with these web-based applications can get an idea of the output from the code (we did this in Smith & Schneider, 2020 and I think it did help with engagement to have people play & share the app they'd built online). This is described in the discussion as "There is also the more complex option to host the web app remotely and pay for more powerful computing infrastructure (e.g. Google Cloud)" but this complexity is all on the developer side, and it makes it much easier for the user (who may not understand Python/GitHub and may not be able or permitted to install onto their machine). The paper also mentions 'Web-apps' a large number of times and then therefore its surprising there wasn't more discussion of how to achieve this.

Yes we agree this is a good idea. We have deployed the web app via streamlit community cloud. It can be accessed here: https://simpy-app-tutorial.streamlit.app/. We have added the link to opening paragraph of the 'Convert to an interactive web app' section of manuscript. Please note that streamlit cloud will put the app to sleep if it is not used for period of time. If this is the case, there will be a short delay while the app 'wakes up'.

R2: Related to the above, there could be much more discussion about the best way to share the interactive model with potential users. Some options are listed, but there are trade-offs involved, ...

We are actively working in this area. We have attempted to include a short summary of issues in the manuscript for you, but there are a lot of things to consider here and it steps outside of the scope/core message of the tutorial. We have added some discussion and citation of relevant *works in the limitations section of the manuscript.*

We thought the best approach to handle your points was to expand the limitations section of the manuscript. We have also signposted readers to your existing material. The following text has been added:

- "Our tutorial has a number of limitations. One issue we do not consider are projects where data, used to derive parameters or the parameters themselves used by the simulation model, may need to be kept private due to governance or ethical reasons. In these instances hosting a model interface on a service such as streamlit community cloud is more challenging. This is a contemporary problem for the UK NHS. Initiatives such as OpenSafely have emerged to tackle reproducible pipelines and could support deriving model parameters (Williamson et al, 2020). There is likely much can be learnt from the Health Economic Evaluation community and research in R where models access secure data and parameters remotely using a secure API (Smith et al, 2022)."

R2: ... especially in relation to data-sharing, something that we tried to solve in (Smith, Schneider & Mohammed., 2023) https://wellcomeopenresearch.org/articles/7-194 using APIs to send models to data, rather than data being sent to models. I don't think we've really cracked it, or explained it particularly well to those who are not familiar with such methods. Therefore we are still exploring different solutions, as this is probably the biggest barrier to modelling transparency in our industry (and I suspect it is the same in healthcare OR). I would be keen to hear the authors' opinions & experiences in Healthcare Operational Research settings and potential solutions.

Regarding model sharing when data are sensitive or commercial in nature. We think your approach of consultant accessing secure data via an API is an important contribution and likely to be highly relevant in the UK with the new NHS Secure Data Environments that are shortly coming online. We are working in a complementary area, but with a slightly different approach (at a less advanced stage) where we are looking to build models that may rely on private data, but will be run by the NHS as opposed to a consultant.

In summary, we have been thinking about this in terms of containerisation plus continuous integration and deployment pipelines. Apologies if you are already familiar with the terminology that we explain below. In summary, a container is a way to wrap up a deployable version of a Linux OS, system tools, FOSS simulation software, a DES model, and an browser based interface. A container registry (e.g. DockerHub) is an online repository that hosts the container image. Using containerisation software (e.g. podman or docker) this can then be pulled to a local machine (e.g. a laptop or server within an NHS Trust) and parameterised using private data - it does not matter what OS the user has installed on their local machine. We used continuous integration through a service like GitHub as it allows us to submit updates to code/models and automatically run a set of unit tests to check for bugs we may have introduced in the new changes. If these tests are passed, we automatically build and push a new version of the docker image to the container registry. We have a book chapter of the general approach accepted for publication (due out next year). We have a pre-review copy of the chapter available as a pre-print https://osf.io/qez45/. This also contains a worked example and a containerised model you can try (you would need to install docker). When compared to your API approach a limitation of our method is that a user must *"pull" updates. For the moment we adopted a "release" approach to continuous deployment so that we can talk to users without confusion e.g. model version 1.0.0 versus model 1.0.1 (if a patch was issued for a bug). But we believe (but have not tested) that we can automate the update of the container with the latest image via a simple start-up script. The benefit is that we can easily provide accessible models with interfaces similar to those in this article, that we know will run on any machine, with very minor updates to our workflow.*

We would be happy to collaborate in this area.

R2: There is no discussion around documentation and testing of code. While I appreciate this is not the primary focus of the paper, if the authors are advocating for sharing the code via GitHub and users deploying the apps themselves (e.g. locally in on organization servers) then model documentation & testing will be crucial (this is perhaps less crucial when the original authors deploy the app themselves - although remains best practice). We have just written a paper on building R packages for health economic models (Smith, Mohammed & Schneider, 2023) https://wellcomeopenresearch.org/articles/8-419 with the aim being that each model should be documented and tested by the original authors in a systematic way before being shared with external parties, I assume there is a similar expectation in OR and with models developed in Python?

Yes we would consider it to be very important. From our previous review work (Monks and Harper, 2023) we know that that published DES models in health rarely have any instructions to run them. Our additional discussion in the limitations section of the manuscript now covers this point.

- "While we have discussed creating model interfaces and wrapper functions for Python DES models, we have not been prescriptive about how these models are logically structured internally or what else should be included. Others have argued that standardisation of code and its organisation will benefit transparency and reuse in Health Economic Evaluation (Alarid-Escudero et al, 2019; Smith et al., 2023). This includes naming conventions, unit testing, and project documentation. The inclusion of comprehensive documentation has been identified as a primary motivator of reuse of code (den Eynden, 2016)."

R2: "streamlit’s community cloud deployment will limit model execution to a single CPU." presumably this is just an issue with streamlit rather than a fundamental problem with web-based models. For example models deployed on AWS/Google Cloud can be run on multiple CPUs.

Yes that is correct. We have modified the text in the strength's section of the manuscript to the following

- "However, in general these benefits are unlikely to transfer to web based models without using a paid tier offered by a web app hosting provider. For example, streamlit's community cloud or ShinyApps.io free tier deployment will limit model execution to a single CPU"

R2: There is not much discussion around the customizability of the UI. Again, I accept

this is not the primary focus of the paper but my experience is that organizations will require the user-interface to look and feel a certain way (often to match their brand) and therefore the feasibility of this using Streamlit may limit its uptake.

This is a good point to raise and one that will be important to NHS organisations. The short answer is yes, streamlit app appearance is very customisable (particularly compared to commercial simulation software). At this time we have not expanded our tutorial material to cover this point. We thought the best way to handle this point was to list it as a limitation of our tutorial and note the potential in this area. We have included the following text:

- "Our tutorial focused on the mechanics of building a usable streamlit interface for DES models, but we have not described detailed customisation options for interfaces. These might, for instance, be important for models where NHS or research funder branding are important. Basic options provided by streamlit include the ability to display NHS or funder logos using the st.image() function. More flexible and detailed control of appearance can be accessed via streamlit's theming options, e.g., controlling font, and background colour of different sections or pages."

References

1. Monks, T & Harper, A. (2023). Computer model and code sharing practices in healthcare discrete-event simulation: a systematic scoping review, Journal of Simulation, DOI: 10.1080/17477778.2023.2260772

2, Harper, A., Monks, T., & Manzi, S. (2023, October 10). Deploying Healthcare Simulation Models Using Containerization and Continuous Integration. https://doi.org/10.31219/osf.io/qez45

Competing Interests: None

Reviewer Report 11 October 2023

https://doi.org/10.3310/nihropenres.14611.r30522

© 2023 Palmer G. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✓ **Geraint Palmer**
Cardiff University, Cardiff, Wales, UK

This article provides a tutorial on how to use streamlit to turn discrete event simulation scripts written in Python into user friendly apps. The reason for this is to help facilitate open simulation software by removing a key drawback to code-based simulations, namely the need to understand the code in order to understand and manipulate the software. In particular it describes how to structure simulation code to be able to integrate nicely with streamlit.

This is a very good paper on an important topic, and highlights one tool that can be useful in facilitating open and accessible software. I learnt a lot here, and have only some minor comments on the manuscript.

- For the code listings I recommend following following the PEP8 style guide (https://peps.python.org/pep-0008/). An easy way to do this is to run the script through Black, a Python linter (https://black.readthedocs.io/en/stable/). This will make the listings much more readable and readily accessible to those familiar with Python.

- The "Methods" section on page 6 has a nice description of 'swap-able' code segments. Can you also add, that in the language of software development, this would be called 'modular' code.

- In Step 5 on page 16, a description of 'set_page_config' is given, but this does not appear in the listings. Similarly, line numbers of not given for 'st.columns(2)', only placeholders.

- On page 11 under "Code to test the model", it would be good to also mention that automated testing of code, such as unittesting, is standard practice in software development (https://software.ac.uk/resources/guides/testing-your-software) and can be useful here too for simulation verification.

- Listing 1 is pseudocode, but written and formatted as Python code, which may be confusing to readers following the tutorial. Can this be formatted differently?

- I am not sure the section on "Simulation Model Logic" on page 8 contributes to the understanding of the overall methodology and is needed. Those unfamiliar with Simpy might still not understand after reading this, and this is not a paper on Simpy.

These are minor comments and overall this is a very useful article.

Is the rationale for developing the new method (or application) clearly explained?
Yes

Is the description of the method technically sound?
Yes

Are sufficient details provided to allow replication of the method development and its use by others?
Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Operational research, simulation, Python

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 15 Dec 2023

Thomas Monks

Response to Reviewer 1 (R1)

R1 This is a very good paper on an important topic, and highlights one tool that can be useful in facilitating open and accessible software. I learnt a lot here, and have only some minor comments on the manuscript.

Thank you for your review and feedback. We are very grateful to you for giving experience and time to improve the manuscript. We are also glad that you learnt something new in our material.

R1: For the code listings I recommend following following the PEP8 style guide (https://peps.python.org/pep-0008/). An easy way to do this is to run the script through Black, a Python linter (https://black.readthedocs.io/en/stable/). This will make the listings much more readable and readily accessible to those familiar with Python.

Thank you for your advise to use Black. This is an excellent piece of software. We confirm that we have done as requested. Please note that there are instances where we have deviated slightly from Black's reformatting in order to better fit in with the journal's formatting requirements. For example, in Listing 5 the Black recommended that Line 26 was not split (it is only 70 characters in length). This is because the journal article cannot accommodate the full length of the line and wraps to a unlabelled line below. Instead we introduced a manual split in the article. We checked all changes from Black auto-formatting with Flake8 to make sure we did not introduce new PEP8 violations. We also checked and updated references to listing line numbers in the main text where needed.

R1: The "Methods" section on page 6 has a nice description of 'swap-able' code segments. Can you also add, that in the language of software development, this would be called 'modular' code

As requested we have described the code as modular.

R1: In Step 5 on page 16, a description of 'set_page_config' is given, but this does not appear in the listings. Similarly, line numbers of not given for 'st.columns(2)', only placeholders.

Thank you for identifying this mistake. We have removed the reference to `set_page_config` as we only intended to show an extract of the full listing here. We have also inserted the correct line number for `st.columns(2)`

R1: On page 11 under "Code to test the model", it would be good to also mention that automated testing of code, such as unittesting, is standard practice in software development (https://software.ac.uk/resources/guides/testing-your-software) and can be useful here too for simulation verification.

This is a very good point. To get readers started we have direct them to the Turing Way's section on code testing (that covers different testing strategies) and Onggo and Karatas (2016) that introduces test driven development and unit testing in the context of (agent based) simulation. We have added the following test to the manuscript:

"In a real simulation project it is recommended that more extensive code testing and verification of model logic is conducted. The full breadth of testing strategies available is out of scope for this tutorial. Interested readers are directed to the code testing section of The Turing Way and in an introduction to Test Driven Development and Unit Testing in the context of computer simulation".

References:
- The Turing Way Community. (2022). The Turing Way: A handbook for reproducible, ethical and collaborative research. Zenodo. doi: 10.5281/zenodo.3233853.
- Bhakti Stephan Onggo, Mumtaz Karatas (2016). Test-driven simulation modelling: A case study using agent-based maritime search-operation simulation, European Journal of Operational Research, Volume 254, Issue 2, https://doi.org/10.1016/j.ejor.2016.03.050.

R1: Listing 1 is pseudocode, but written and formatted as Python code, which may be confusing to readers following the tutorial. Can this be formatted differently?

This is a very good point. We agree that Listing 1 can be improved. We have modified Listing 1 to use more standard psuedocode notation such as SUBROUTINE and left arrows for variable assignment.

R1: I am not sure the section on "Simulation Model Logic" on page 8 contributes to the understanding of the overall methodology and is needed. Those unfamiliar with Simpy might still not understand after reading this, and this is not a paper on Simpy

Thank you for raising this point. We have considered this point and decided that we would like to keep the section in the manuscript. The reason is that we would like to be able to clearly sign-post the reader to our Extended Material (and external website) where we have additional tutorial material that can be used to recreate and understand the simulation model implementation. We agree that the papers focus is not simpy, but we would like to keep a clear audit trail for reproducibility.

Competing Interests: None.