
Article:
Smith, R., Mohammed, W. and Schneider, P. (2023) Packaging cost-effectiveness models in R: a tutorial. Wellcome Open Research, 8. 419. ISSN: 2398-502X

https://doi.org/10.12688/wellcomeopenres.19656.1

METHOD ARTICLE

Packaging cost-effectiveness models in R: A tutorial. [version 1; peer review: 2 approved with reservations]

Robert Smith [1,2], Wael Mohammed [1,2], Paul Schneider [1,2]

[1]ScHARR, The University of Sheffield, Sheffield, England, S1 4DA, UK
[2]Dark Peak Analytics, Sheffield, UK

v1 **First published:** 21 Sep 2023, **8**:419
https://doi.org/10.12688/wellcomeopenres.19656.1
Latest published: 21 Sep 2023, **8**:419
https://doi.org/10.12688/wellcomeopenres.19656.1

Abstract

Background: The use of programming languages such as R in health economics and decision science is increasing, and brings numerous benefits including increasing model development efficiency, improving transparency, and reducing human error. However, there is limited guidance on how to best develop models using R. So far, no clear consensus has emerged.
Methods: We present the advantages of creating health economic models as R packages - structured collections of functions, data sets, tests, and documentation. Assuming an intermediate understanding of R, we provide a tutorial to demonstrate how to construct a basic R package for health economic evaluation. All source code used in or referenced by this paper is available under an open-source licence.
Case Study: We use the Sick Sicker Model as a case study applying the steps from the tutorial to standardise model development, documentation and aid review. This can improve the distribution of code, thereby streamlining model development, and improving methods in health economic evaluation.
Conclusion: R packages offer a valuable framework for enhancing the quality and transparency of health economic evaluation models. Embracing better, more standardised software development practices, while fostering a collaborative culture, has the potential to significantly improve the quality of health economic models, and, ultimately, support better decision making in healthcare.

Keywords

HTA, R, Open-source, Health Economics

Open Peer Review

Approval Status ? ?

	1	2
version 1 21 Sep 2023	? view	? view

1. **Isaac Corro Ramos**, Erasmus University Rotterdam, Rotterdam, The Netherlands

2. **Joe Moss**, York Health Economics Consortium, York, UK

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

Corresponding author: Robert Smith (rasmith3@sheffield.ac.uk)

Author roles: Smith R: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Project Administration, Resources, Software, Supervision, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; **Mohammed W**: Formal Analysis, Investigation, Methodology, Software, Validation, Writing – Review & Editing; **Schneider P**: Formal Analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – Review & Editing

Competing interests: R.S. Is part of the Scientific Committee for R for HTA, an academic consortium whose main objective is to explore the use of R for cost-effectiveness analysis. R.S., P.S. and W.M. work for Dark Peak Analytics Ltd; a company specialising in the application of techniques similar to those discussed in this paper in epidemiology and health economic evaluation.

Grant information: R.S., W.M and P.S. are joint funded by the Wellcome Trust Doctoral Training Centre in Public Health Economics and Decision Science [108903] and the University of Sheffield.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Smith R, Mohammed W and Schneider P. **Packaging cost-effectiveness models in R: A tutorial. [version 1; peer review: 2 approved with reservations]** Wellcome Open Research 2023, **8**:419 https://doi.org/10.12688/wellcomeopenres.19656.1

First published: 21 Sep 2023, **8**:419 https://doi.org/10.12688/wellcomeopenres.19656.1

Introduction

Health economic models are increasingly used to inform decisions about the allocation of resources in healthcare systems and other government departments in an attempt to improve population health[1,2]. It is imperative that these models are robust, transparent and efficient to maintain and develop. At the moment, building models in spreadsheet software is standard practice[3]. However, with the increasing complexity of economic evaluations and models, the use of programming languages, especially R (RRID:SCR_001905) due to its strong statistical analysis functionality and popularity in biomedical research, is becoming more popular. Programming languages like R offers numerous benefits over spreadsheet software in terms of reducing errors, improving transparency, and facilitating collaboration among health economists[4].

However, in the absence of clear guidance, many health economists find themselves devising their own structures for model development in R. The lack of standardisation has led to a proliferation of different coding styles, making it difficult to share model code, review, or replicate models, and to facilitate collaboration Alarid-Escudero *et al.*[5]. Without a sound code structure, models may become difficult to debug, modify, and extend. In addition, the lack of agreed upon standards leads to a lack of consistency in model development and reporting which can lead to confusion and misinterpretation. Furthermore, it can exacerbate the reluctance to share code among researchers, as found by Emerson *et al.*[6] who report that many health economists do "not want to confront the issue of publishing their source code, or at the very least, may not view source code publication as a priority" (p.1410). This may be due to apprehension about errors, or because of a reluctance to share code regarded as 'messy' or not conforming to other researchers' standards.

R packages provide a standardised approach for model development. packages serve as modular extensions that enhance the capabilities of the base R software by providing functions, data sets, and documentation[7]. They can be developed for internal use within an organisation or shared as open-source resources for the wider community, for example via the Comprehensive R Archive Network (CRAN) or GitHub[8,9]. The modular nature of R packages enables scalable and reproducible health economic evaluation models, which ultimately benefits the health economics community as a whole and would help to facilitate a growing demand for more transparent open-source modelling[10]. Several packages exist to provide generic functions for a range of health economic model types - notable examples are the heemod and hesim packages[11,12]. These packages do not contain a health economic model, but are instead a set of tools to help health economists develop their own specific models. The focus of this paper is on a framework and methods required to build an R package for a single health economic evaluation model.

A previous paper by Alarid-Escudero *et al.*, 2019[5] outlined a proposed health economic evaluation model structure in package format, outlining the benefits of a standardised coding framework. This paper adds to this work, by providing a tutorial on the process of building a custom package from scratch, rather than using a template. We also provide a case study where we adapt the 'Sick Sicker' model, originally developed by Krijkamp *et al.*, 2018[13], into a package with a different structure to Alarid-Escudero *et al.*, 2019[5] but based on similar principles. We have previously used this same model to demonstrate the value of web-based user-interfaces[14] to make models more usable and application programming interfaces and automation[15] to reduce data sharing requirements and move towards Living Health Technology Assessment (HTA).

The methods section of this paper includes both a justification for the use of Health Economic Model packages and a tutorial on building a simple R package for a model. The results section includes a case study in which a health economic model commonly used for teaching was built into an R package. The paper concludes by discussing potential avenues for package validation, and in particular the role that trusted experts and institutions can play in endorsing certain packages, with the goal of improving efficiency, quality, and transparency in the field of health economics and decision science.

Methods

This section has two main parts. The first part discusses health economic models in R, and the advantages of using packages for the health economist, model reviewers and the wider research community. The second part guides the reader through the basics of building an R package for health economic evaluation code.

The advantages of Health Economic Evaluation model Packages

At its simplest, a health economic evaluation model can be thought of as an algorithm which takes a set of inputs, for example parameter inputs or even individual patient-level data, and returns some results, for example total discounted costs and quality-adjusted life years (QALYs). A schematic can be seen in Figure 1 below.

The types of models built for health economic evaluation vary in complexity from simple decision trees to agent based models[16]. Except for the simplest models, which are typically not programmed in R, the code base is typically larger than for descriptive analysis or causal inference, requiring a large number of calculations which are interlinked. In R, best practice is to modularise the model into a set of functions, each of which can be defined and tested separately[17]. These functions can be used in sequence and/or nested inside one another, as shown in Figure 2. The entire model can itself be a single wrapper function which performs all of the steps for a given set of parameters and returns a set of results and/or figures and tables. This has previously been outlined by Alarid-Escudero *et al.*[5] who advocate the construction of the model as a single function which "facilitates subsequent components of model development and analysis, as these processes will all call the same model function but pass different parameter values and/or calculate different final

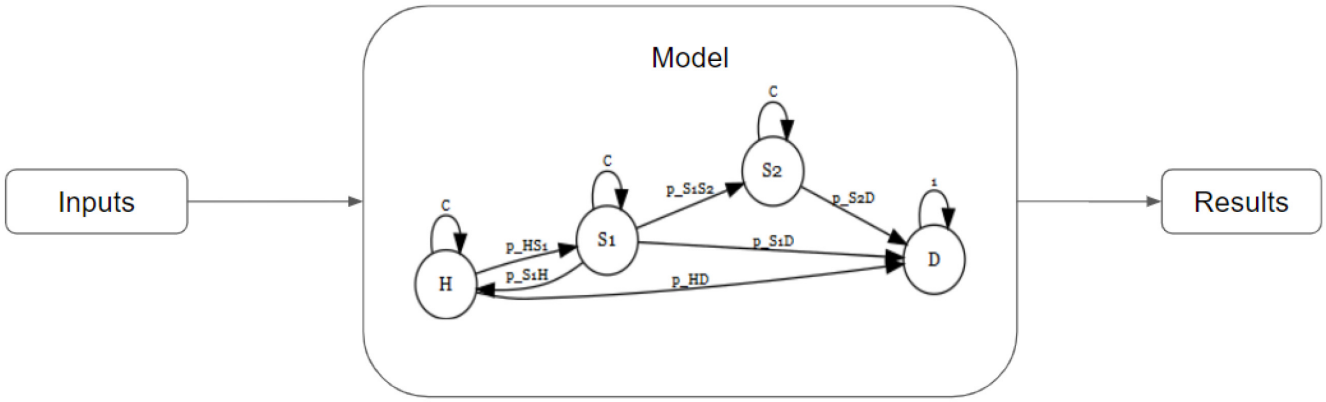

Figure 1. Health Economic Model as an algorithm taking inputs and returning results.

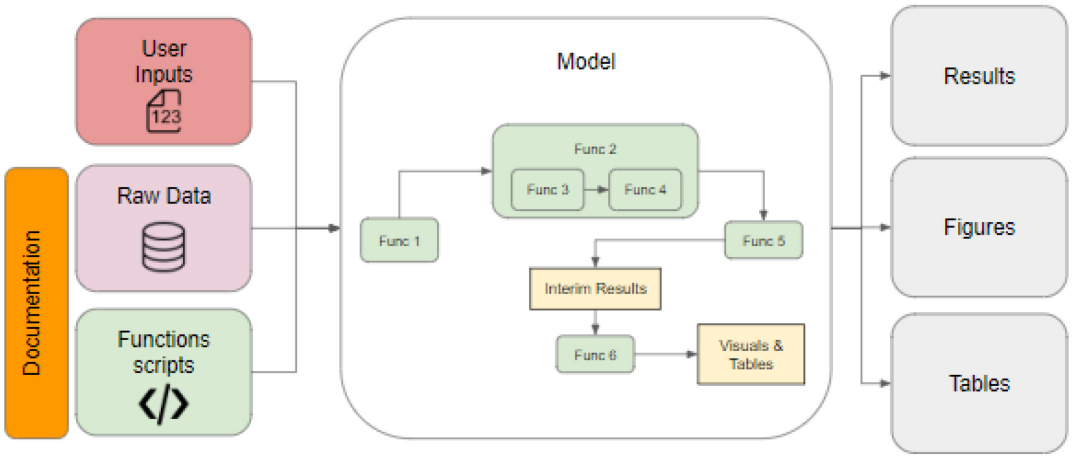

Figure 2. Schematic of a typical model structure taking raw data and user inputs and using a set of functions to return results in the form of data, publication tables, and figures.

outcomes from the model outputs" (p.1332) and has been the way in which we have interacted with models via application programming interfaces (APIs) and web-interfaces[14,15].

There is limited guidance on how to structure a health economic model built in R, although efforts are underway by regulatory agencies, including the National Institute for Health and Care Excellence (NICE) and the Dutch National Health Care Institute (ZIN), to identify best practices for submissions including health economic models built using R. A recent paper by Alarid-Escudero *et al.*[5] provides one framework, which some health economists have since used[18,19], but there remains no consensus. However, in our experience as a minimum in a well constructed R model, functions tend to be stored in a folder (generally "R/"), with other folders containing the unit tests (checks to ensure functions work as intended, as described in 'Unit Testing' below) and the data (described in 'Data' below) required by the model. There should also be documentation describing the overall modelling approach and what each of the individual functions does, and there is generally at least one folder for outputs such as results data, figures or tables. Our previous work on automated reporting has given an example folder structure where there is an automated Rmarkdown/Quarto report included as in Smith *et al.*, 2022[15]. In the Alarid-Escudero *et al.* framework there are four separate output folders (Tables, Figures, Report, Outputs). Figure 3 shows a minimal example of a folder structure for a model built in R, appreciating that more complex models may require more subfolders, for example containing model objects.

Those readers with experience developing Packages in R will notice that the folder structure shown in Figure 3 above is very similar to that of an R Package as shown in Figure 4 below. This is not surprising since the R package structure has emerged organically within the R software development community as a method of storing, documenting and sharing code. Since good practice in health economic evaluation modelling in R results in a folder structure that is very similar

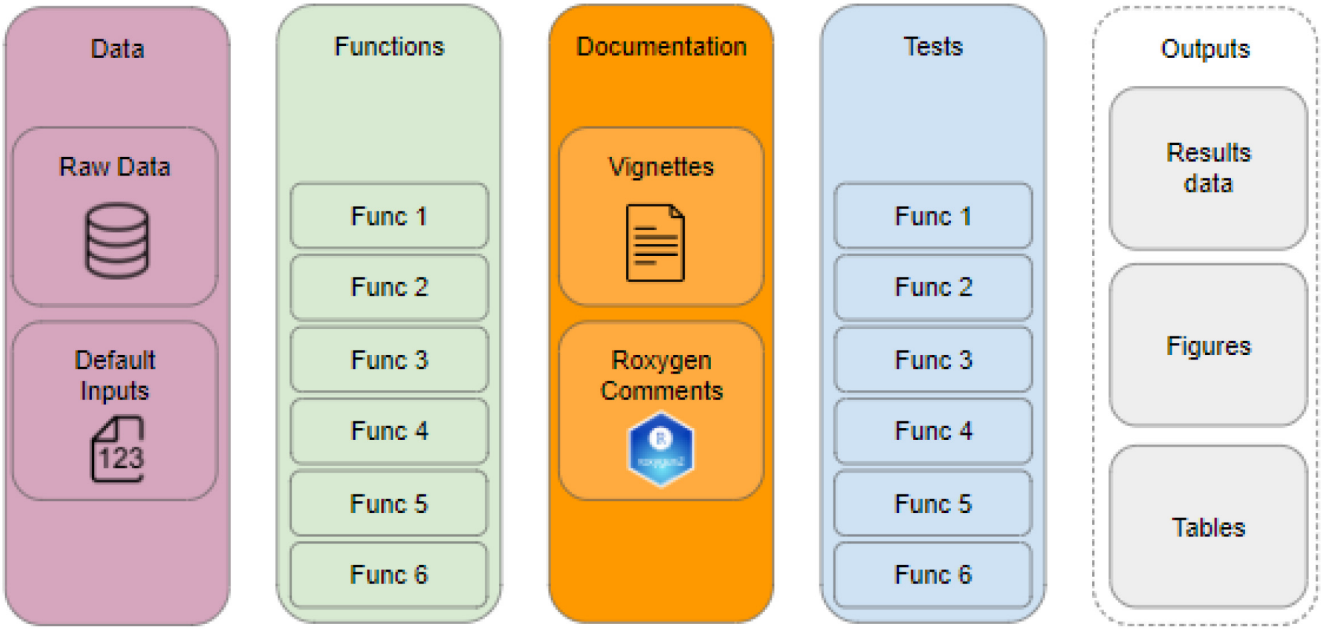

Figure 3. A minimal example of a folder structure for a model built in R.

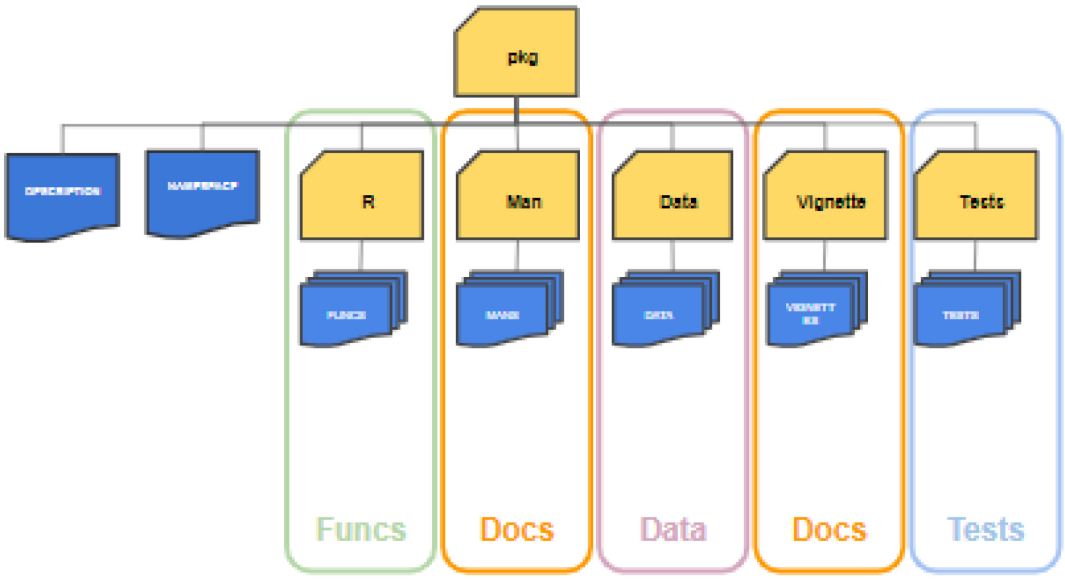

Figure 4. Basic R Package folder structure including tests and vignettes categorised by type of file - either Functions, Documents, Data or Tests.

to an existing framework for software development in other industries, health economic modellers should use the existing Package structure, as outlined in Wickham[7], as standard and add additional folders (e.g. in the 'Inst' folder) if they are required.

In addition to the benefits of standardisation, ease of testing, and documentation, using packages makes it easier for methods (functions) from one model to be used by others. This can help to ease convergence in modelling methods, improve model building efficiency, and make review much easier. Those familiar to R will have previously installed packages from CRAN or GitHub, for example using the `install.packages()` function provided in base R. By storing health economic evaluation model code on software development platforms like GitHub, it is easy for others to install the code to use specific functions from one model in another. Figure 5 below shows

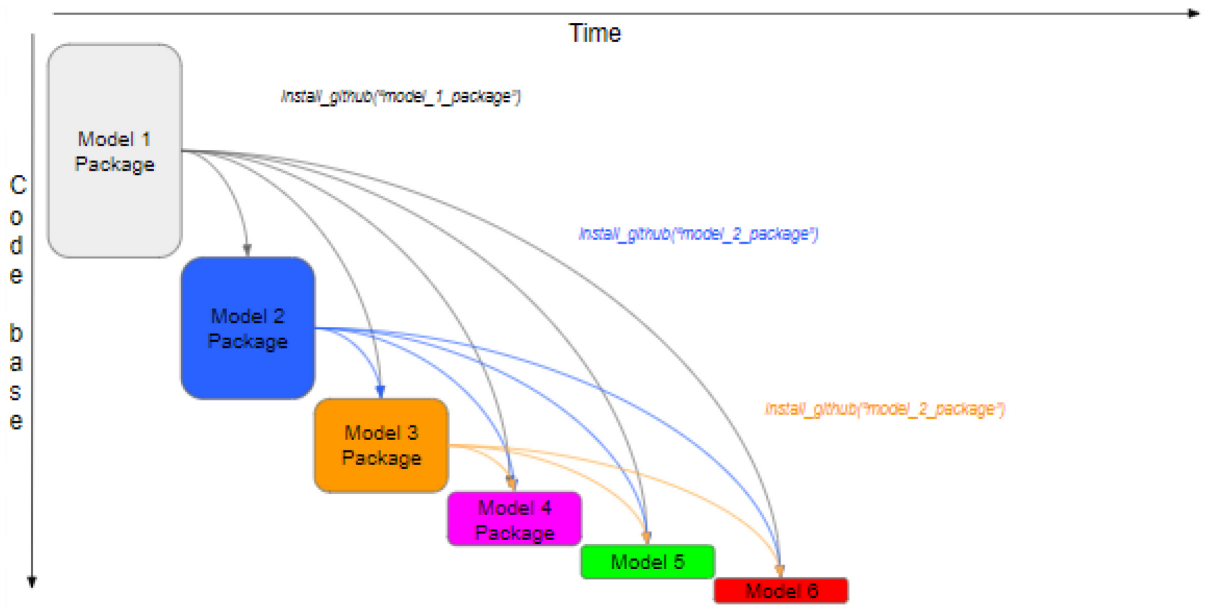

Figure 5. Development of the health economic modelling code base over time, as 6 model packages are published sequentially.

how the code from the Model-1-Package could be used by the Model-2-Package, Model-1-Package and Model-2-Package to Model-3-Package and so on, significantly reducing model development time and requiring the functions which are used multiple times to be reviewed only once, reducing duplication of model review processes. The overall result is that subsequent models have a smaller marginal effect on the code-base since they utilise existing functions where possible.

Over time, some functions within the model packages may be reviewed by a trusted body, and collated into a one or more packages which bring together useful and related functions (as shown in Figure 6). Some functions may be very generic, such as the calcICER function, whereas other functions may be field specific (e.g. testing functions for an infectious disease cost-effectiveness model). This would be a large and ongoing project, but one that could be undertaken by relatively junior software engineers and overseen by health economists. We believe that the benefits, in terms of the reduction in the costs of developing and reviewing models over the long run, would massively outweigh the relatively small immediate costs of this exercise - since most of the coding will have already been done in the individual models.

This has significant benefits for regulators, and those tasked with reviewing health economic evaluation models, such as Evidence Review Groups (ERGs) tasked by NICE to produce a review of the economic evaluation model for a UK submission. When reviewing steps of a model which use functions directly from the validated package, reviewers may have more confidence that the function is working correctly. While reviewers will still have to ensure that functions are applied correctly, the increased confidence should significantly reduce review time. The result is that future models will tend to use the validated package since it is less likely to result in negative feedback from reviewers. Figure 7 shows the development of Model 7-9 which uses the regulator preferred package simultaneously with little additional code development.

This process would continue iteratively, with improvements and additions to the validated package made on an ongoing basis as part of an open-source ecosystem as suggested by Dasbach and Elbasha[17].

To summarise, health economic evaluation models can be thought of as algorithms that take inputs such as raw data and parameters, and return results like total discounted costs and QALYs. While these models range in complexity from simple decision trees to microsimulation or agent-based models, there is a general movement towards increased complexity and the use of script based programming languages like R. As R increases in popularity it is important to have a shared framework to standardise model structure. An existing framework for software development already exists in R, the package. By using R packages, health economic modellers can standardise, test, document, and share code more efficiently, potentially leading to the development of regulator-approved or community-validated packages that can reduce model build and review costs and increase quality in the long run. However, this will require a considerable amount of training for health economists who have not previously used R, or are new to working in R and have not built their first package. The remainder of this paper aims to address this gap.

Tutorial: The basics of R Packages for Health Economic Evaluation

In this tutorial, we aim to provide a comprehensive introduction to the fundamental principles of R packages, specifically

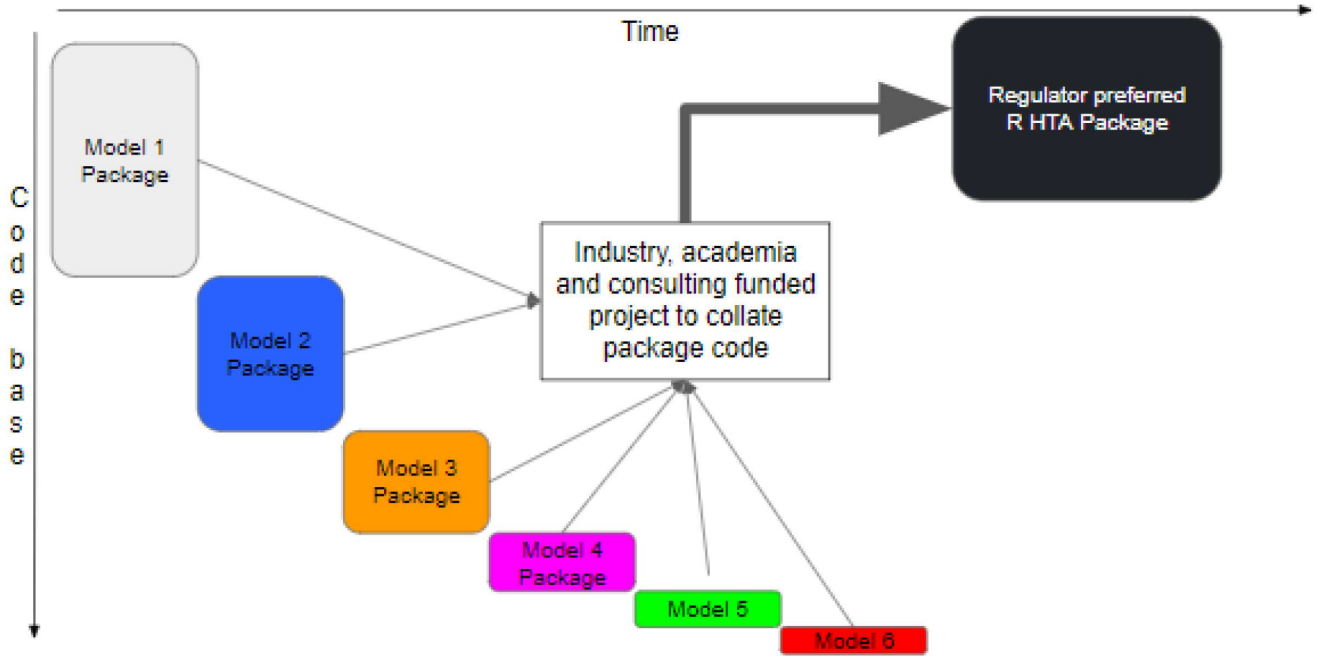

Figure 6. Development of a regulator preferred R Package collated from existing model functions.

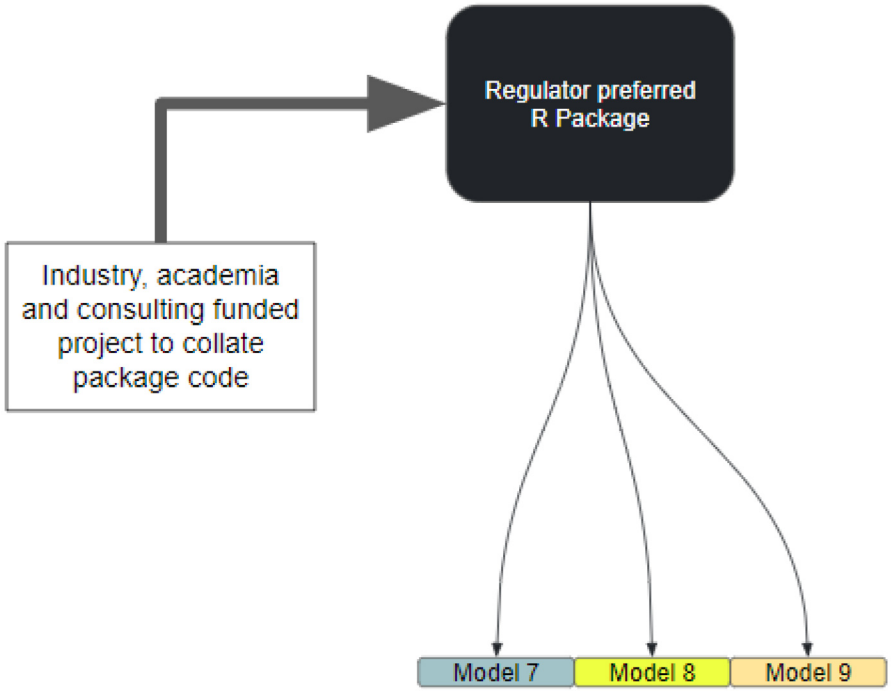

Figure 7. Use of regulator preferred Package by subsequent models with lighter touch review required.

in the context of a health economic evaluation model. We start by introducing the concepts intrinsic to R packages and elucidate their relevance in health economic evaluations. Our tutorial guides readers through a sequential and instructive process that includes several key steps.

Firstly, we explain the basic structure of an R package's skeleton, detailing crucial components such as the DESCRIPTION, NAMESPACE, and R folders and providing a guide on initiating a new package in RStudio (RRID:SCR_000432) utilising the devtools (RRID:SCR_016961)[20] and usethis[21] packages.

Secondly, we delve into the organisation of functions and data within the R package directory, highlighting the importance of code modularity and reusability. Thirdly, we underscore the necessity of comprehensive documentation for facilitating comprehension, communication, and transparency. We introduce the R documentation system that leverages the Roxygen2 package for function documentation generation and discuss the use of vignettes for documenting higher-level concepts, such as structural modelling assumptions[22]. Fourthly, we focus on the significance of unit testing for ensuring the precision and reliability of R packages, presenting the testthat package as a resource for creating and executing unit tests for package functions iteratively[23]. Lastly, we illustrate how to construct (in RStudio) and install an R package locally and explore multiple strategies for distributing packages, including via platforms like CRAN or GitHub[8].

This tutorial demonstrates how to create an R package named HECONpack[24] from scratch. HECONpack contains a single function `calcICER()` that calculates the incremental cost-effectiveness ratio (ICER) from baseline and intervention costs and effects (QALYs). The tutorial shows how to set up the package, use Roxygen2 comments to document the function, create a test suite using testthat and run checks to ensure that HECONpack meets some standard software development rules. It also includes instructions on how to add a license and how to make some data available to users of the HECONpack. It concludes by showing how others could install the package from GitHub to use the function and data for their own models.

This tutorial is targeted for those new to developing R packages, or who would like to brush up on the basics. The content is based on the book R packages: organize, test, document, and share your code by Hadley Wickham[7]. The book is not specific to health economics, but may serve as a useful point of reference. We provide links to specific sections of the open-access online book throughout.

Defining the scope and objectives of the Package. Before creating a package, the objective of the package should be defined, and it should be determined, whether alternative packages already exist. If there is a close alternative it may be more efficient to adapt an existing R package. The adaptations can then be submitted to the original package author or maintainer for inclusion in the original package.

The aim of the package we will create for this tutorial, the HECONpack, is to allow users to calculate an ICER from a set of four numbers, the costs and effects in both the baseline and intervention scenarios. To achieve this, a single function called calcICER will be created. In this example, we assume that no relevant package exists (of course in reality many other packages already have this functionality). More information on when and why to set up an R package can be found here.

Setting up the R Package skeleton. To create a new package, the devtools library is required. It can be installed by running the following command: `install.packages("devtools")`. The devtools package is designed to aid R package development and management, providing useful functions and tools[20]. It relies on the usethis package in the background and will automatically load functions from that package too. To load the devtools library into the R session, execute `library(devtools)`.

Now, a new package can be created using the create_Package function. For example, to create a package named 'HECONpack' the following command is used: `devtools::create_package(path = "HECONpack")`. This command will generate a new R package skeleton in the specified path, creating a project file named "HECONpack.Rproj" in a folder called "HECONpack" at that path. The skeleton provides an empty R package with a basic directory structure and some necessary files. It includes an R folder, a DESCRIPTION and a NAMESPACE file as well as the R project *.Rproj* file. Table 1 shows the structure of the newly created folder.

More information on R projects can be found here and for more guidance on package setup more generally please refer to this link.

Incorporating our first function. Now we have a basic Package structure with an R folder, we can create our first function. To begin, we will create a new R script file called

Table 1. File structure created by devtools as a skeleton.

path	type	description
.Rbuildignore	file	files to ignore when building Package
DESCRIPTION	file	metadata, e.g. name and version.
NAMESPACE	file	from Roxygen, ensures names dependencies etc.
R/	directory	R functions

calcICER.R within the R folder of the directory. This can be done using the code shown below:

```
usethis::use_r("calcICER")
```

In this example we are going to insert a single function into the R script, but for larger packages we may want to store a set of related functions in a single R script. The R code with its accompanying Roxygen code (which used later to document the function in a help file) is provided below. To replicate this step, paste the code for the `calcICER()` function into the newly created file . Finally, save the file.

```
#' Calculate the Incremental Cost
#' Effectiveness Ratio (ICER)
#'
#' Calculates the incremental effect and
#' incremental costs of an intervention
#' compared to baseline and then uses the
#' results to calculate the ICER.
#'
#' @param e_int single value for effect
#' (e.g. Total QALYs) in intervention group.
#' @param e_base single value for effect
#' (e.g. Total QALYs) in base group.
#' @param c_int single value for cost
#' (e.g. Total £) in intervention group.
#' @param c_base single value for cost
#' (e.g. Total £) in base group.
#' @return an single value for the ICER.
#' @importFrom assertthat assert_that
#' @export
#' @examples
#' calcICER(e_int = 28.3,
#'          e_base = 22.5,
#'          c_int = 10000,
#'          c_base = 9200)
calcICER <- function(e_int,
                     e_base,
                     c_int,
                     c_base) {
  # Check that all inputs are numeric
  assertthat::assert_that(
  is.numeric(c(e_int, e_base,
               c_int, c_base)),
  msg = "All inputs must be numeric."
    )
  # calculate incremental costs and effects
  inc_e <- e_int - e_base
  inc_c <- c_int - c_base
  # calculate the ICER
  icer <- inc_c / inc_e
  return(icer)
}
```

As well as comprehensive documentation, it is good practice to insert 'asserts' which check the user specified input parameters to reduce execution errors and erroneous outputs. This is implemented above using the 'assertthat' package in R[25]. The `assertthat::assert_that()` checks that all of the arguments are class numeric. Any violation of these checks will halt the evaluation of the affected function, triggering an informative message, in this case "All inputs must be numeric values or vectors", to help guide the user to fix the problem. These checks should help other users avoid unexpected behaviours or unknown errors.

Once the the calcICER function is added to the "calcICER.R" file, all the package functions can be loaded into the R session for further development by running:

```
devtools::load_all()
```

Alternatively, the RStudio keyboard shortcut Ctrl + Shift + L on Windows, or Cmd + Shift + L on a Mac can be used, to achieve the same effect. This command loads all the functions from the package, enabling the health economist to work with the functions quickly.

Documentation. Good documentation helps users understand the purpose and usage of your package and its functions.

To generate the documentation for the package, the `devtools::document()` function from the devtools library can be executed. Alternatively, the RStudio keyboard shortcut Ctrl + Shift + D on a Windows machine or Cmd + Shift + D on a Mac can be used.

This command will automatically generate the necessary documentation files, stored in the "man" folder. for the package based on the code and 'Roxygen2' tags. These Roxygen tags have been developed in such a way as to standardise code documentation to make it easier to understand what each function in a code base is doing and how they link together. They consist of tags preceded by '@' that describe key elements of the function. Key tags include '@title' (general function purpose), '@description' (longer description of purpose), '@param' (input argument, each in turn), '@return' (outputs) and '@examples' (example usage). These collectively provide a thorough, structured overview of the function's operations, making it easier for users to review, use or adapt the package. More information on documenting using Roxygen can be found in the Roxygen2 R package documentation[26].

Those using the package can see the documentation for any function using the `help()` function in R. For example `help("calcICER")` generates the help file shown in Figure 8 below.

More information on documentation can be found here.

Checks. During the development process, it is useful to regularly check that the R package structure, function documentation, and code conforms to R package development guidelines[20]. To run checks on the package, the `devtools::check()` function from the devtools library can be used. Identified issues

calcICER {HECONpack} R Documentation

Calculate the Incremental Cost Effectivness Ratio

Description

Calculates the incremental effect and incremental costs of an intervention compared to baseline and then uses the results to calculate the ICER.

Usage

```
calcICER(e_int, e_base, c_int, c_base)
```

Arguments

`e_int`	a single numeric value representing the effect (e.g. Total QALYs) in the intervention group.
`e_base`	a single numeric value representing the effect (e.g. Total QALYs) in the base group.
`c_int`	a single numeric value representing the cost (e.g. Total £) in the intervention group.
`c_base`	a single numeric value representing the cost (e.g. Total £) in the base group.

Value

an single numeric value for the ICER.

Examples

```
calcICER(e_int = 28.3, e_base = 22.5, c_int = 10000, c_base = 9200)
```

[Package *HECONpack* version 0.0.0.9000]

Figure 8. Help file for the calcICER function in R.

will be flagged, as errors, warnings, or notes, depending on the severity of the problem. A log file is also generated, and is referenced by any warnings and errors to provide a route by which to investigate further. These issues should be addressed to ensure that the package is functioning correctly and adheres to the best practices. A successful check at this stage is shown in Figure 9.

```
── R CMD check results ──────────────────
Duration: 35.8s

0 errors ✓ | 0 warnings ✓ | 0 notes ✓

R CMD check succeeded
```

Figure 9. HECONpack check output in RStudio.

***Licencing*.** The first time a check is run, it should flag a warning that there is no licence included in the package documentation. Licensing is an important aspect of package development, as it defines the terms under which others can use, modify, and distribute a package. One of the commonly used licences is the MIT licence which allows users to freely use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software, provided that the original copyright notice and disclaimer are included[27].

To add the MIT License to the package (many others available), the `usethis::use_mit_license()` function can be used. This command will create a LICENSE file in the root folder of the package, containing the text of the MIT

License. Additionally, it will update the LICENSE.md file and the "License" field in the DESCRIPTION file of your package, indicating the use of the MIT License.

Description. Now we have a working function that passes all checks, we will add standard meta-data to the DESCRIPTION file contained in the package. The DESCRIPTION file contains essential metadata about the package, such as the package name, version number, author name, and dependencies on other packages. It is written in a specific format that can be read by R and other software.

To update the DESCRIPTION file, located in the root folder of the package, the contents of the file can be changed as needed. Some common fields to update include:

- Package: The name of the package
- Version: The current version number of the Package (e.g., "1.0.0").
- Title: A brief, human-readable description.
- Description: A more detailed description.
- Authors: The package author(s) and their roles.
- Licence: The licence under which the package is distributed (e.g., "MIT").
- Depends or Imports: Any R Packages that the Package depends on, listed with their minimum required version numbers (e.g. dplyr >= 1.0.0).

This information is important for those trying to understand your package and how it fits into the wider health economic evaluation code base. More information on descriptions can be found here.

Namespace. The NAMESPACE file specifies the dependencies of the package and specifies the functions and data which are to be made available to users once the package is loaded. This file is generated automatically from the Roxygen2 comments in the R code files using the `devtools::document()` function. The NAMESPACE file is a crucial component of package development, ensuring that users have access to the intended functions and that internal functions remain hidden. The file should not be edited manually. More information on this file, and what it does, can be found here.

Unit testing. Unit testing is a method of software testing where individual units or components of a software system are tested to determine if they are fit for use. Unit tests are conducted to verify that the code is operating as expected and identify any bugs or errors that may have been introduced during the development process. Writing unit tests is an essential aspect of package development, as it ensures stability of the functions, even as developers make changes to the code[7].

The code below creates a testing file structure and some example tests to insert. It starts by using the `usethis::use_test()` function to create a new test file for your function. This command will create a new test file named "test-calcICER.R" within the "tests/testthat" folder of HECONpack. The test cases can be inserted in the newly created file using `testthat::test_that()`. Two example tests are included in the chunk below.

These tests check if the calcICER function returns the expected results when given simple integer inputs and vector inputs, respectively.

```
test_that("Simple integers work", {
  expect_equal(
  calcICER(e_int = 10,
           e_base = 0,
           c_int = 20,
           c_base = 10),
           expected = 1)
})
test_that("Vectors work", {
  expect_equal(
  calcICER(e_int = c(1,2,3),
           e_base = 0,
           c_int = 10000 * c(2,3,4),
           c_base = 0),
  expected = (10000 * c(2,3,4)) / c(1,2,3)
  )
```

The following command runs all tests within the package's "tests/testthat" folder:

Alternatively, the RStudio keyboard shortcut Ctrl + Shift + T on a Windows machine or Cmd + Shift + T on a Mac can be used to run the tests.

The test results are displayed in the R console as below shown in Figure 10, providing information on which tests passed and which tests failed, or if any warnings were generated. This information can be used to identify issues in the functions and to make the necessary changes to the code.

Running tests is a best practice in software development, providing confidence in the quality of the code and allowing to catch and fix issues early.

```
ℹ Testing HECONpack
✓ | F W S  OK | Context
✓ |          2 | calcICER

══ Results ═══════════════════════════════
Duration: 0.3 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 2 ]
```

Figure 10. Testing output for single test file which includes two tests which pass.

More information on testing can be found here.

Data. The following steps show how to include example data in the package. Firstly the example dataset is prepared as an R object. In the code below, we create a data frame called 'df_res_example'. We then use the `usethis::use_data()` function to include the example dataset in HECONpack, as shown below.

```
df_res_example <- data.frame(
    e_int = runif(n = 1000,
                  min = 200,
                  max = 1000),
    e_base= runif(n = 1000,
                  min = 0,
                  max = 500),
    c_int = runif(n = 1000,
                  min = 10000,
                  max = 100500),
    c_base = runif(n = 1000,
                   min = 5000,
                   max = 40000)
     )
usethis::use_data(df_res_example)
```

Note that the provided data will be available to users of the Package.

Vignettes. A vignette is a long-form guide that provides a comprehensive overview of the package, its functions, and its usage. Vignettes are an excellent way to help users understand how to work with the package effectively.

The code below creates a vignette skeleton for HECONpack, which can then be edited.

```
usethis::use_vignette("my-model-desription")
```

This command will create a new vignette file named "mymodel-description.Rmd" within the "vignettes" folder of the package. This R Markdown file serves as a vignette template, which can be customised as needed. The vignette could provide a comprehensive overview of the package, its functions, and its usage, including examples and explanations. It is possible to have multiple vignettes looking at different functionality. When users build HECONpack it will be included in the final package distribution.

In the long run, we could foresee the vignette being a complement to the technical health economic evaluation report, which would be reviewed by an external reviewer, for example an Economic Research Group at a University. The combination of unit tests testing specific functions, and a technical report vignette, defending methods used, should improve the quality of submissions and reduce the time taken to review them.

Building and Installing. After completing all steps described above, the project directory folder for HECONpack includes several files and folders – see Table 2 below. To install the package to make it available from within other projects (but on the same computer), users can run:

```
devtools::install()
```

This then enables the user to call `library("HECONpack")` to load the package functions and data as they would with any other installed package. To disseminate the package to others, alternative solutions are required.

Dissemination. There are multiple ways in which R packages can be shared with others. The Comprehensive R Archive Network (CRAN) is the official repository for R packages, and it is the primary source from which most R uses download packages. The CRAN Repository Policy and the Checklist for CRAN submissions (r-project.org) describe the process of submitting a package to CRAN.

The required formatting and preparing of a package for publication can be time consuming. Alternatively (or in addition), Packages can also be made available as code repositories, hosted on software development platforms such as GitHub. Hosting in this way is generally much simpler than the CRAN submission process, and allows sharing of developmental versions of packages.

The devtools package enables anyone with an internet connection to install any R package contained in an open-source repository on GitHub using the function devtools::install_github("account/package"). The code below shows how to install HECONpack. The code first removes the 'HECONpack'

Table 2. File structure created by the end of this session.

path	type	description
inst/	directory	installed files when user installs Package
man/	directory	md files documenting for functions
data/	directory	data available within Package
vignettes/	directory	generally used to showcase Package functionality
tests/	directory	unit tests designed to ensure code works as intended

package if it is already installed (which it would be, following the steps above). It then installs HECONpack from the specified GitHub repository, in this case 'dark-peak-analytics/HECONpack'.

```
remove.packages(pkgs="HECONpack")
devtools::install_github(
    "dark-peak-analytics/HECONpack"
    )
```

The version of the package on GitHub will then be installed, and its functions will be available for use. This method of installation is more convenient and user friendly, encouraging more users to explore and adopt the package. To submit changes to the functionality of a package, a, so called, pull request can be submitted to the HECONpack repository maintainer. This facilitates a structured and transparent process, by which developers from the community can suggest changes to existing code repositories.

The process described here would be repeated iteratively as the model is developed. There is no obligation to make code open-source, the process could be conducted internally and shared with specified individuals only, for example via private GitHub repositories. Similarly, some components of the model can be shared but not others. APIs could be used to keep particularly sensitive algorithms confidential while allowing users to obtain results from inputs provided[15], and there is no requirement to share data in the package - sensitive data could be accessed from remote servers or provided separately to a local machine. Dummy data could be included as an internal dataset within the package to allow users to interrogate the model code.

Case study

We demonstrate the concepts introduced in the methods section in a case study by using converting the well known Sick-Sicker model into a standalone R Package, 'sicksickerPack'. More details on the model can be found in Alarid-Escudero *et al.*[28], Krijkamp *et al.*[13], but briefly, the Sick-Sicker model creates a simulation involving a hypothetical cohort of healthy (H) people susceptible to a disease with two stages of illness, 'Sick' (S1) and 'Sicker' (S2). Those with the illness are subject to an elevated risk of death and a decline in quality of life (QoL) compared to their healthy counterparts. The model simulates the cost-effectiveness of a hypothetical treatment which enhances the QoL for those in S1 but has no impact on the QoL for those in the S2 state.

This case study aims to showcase how a health economist can 'package' a decision-model analytic model. We compare the packaged model, available at https://github.com/dark-peak-analytics/sicksickerPack to the version described in previous publications[14,15,28], briefly explaining both the process of building the R package and the functionality of the resulting package.

The original model script contains the definition of the model parameters transition matrix, the Markov trace estimation and the calculation of the cost-effectiveness outcomes. In this paper we aim to achieve the same goal as the published model script and, as discussed earlier, to make it easier to review, debug, improve, and reuse our code (functions) in developing other decision-analytic models in R.

The package code is contained in several folders. As in the tutorial above, R functions are contained in 'R/', documentation in 'man/' and unit tests in 'tests/'. Dummy data, to be used by those without access to sensitive data is contained in 'data/'. A 'DOCUMENT' file contains meta data about the package. Figure 11 contains a schematic showing how the sicksickerPack R Package works.

A user of the model would need to run `run_sick-Sicker_model()`. This wrapper function is specific to the Sick-Sicker model. When called, the function either uses data specified by the user - for example dummy data that is included as part of the package, Or fetches model parameters from a remote server or API using `get_model_params()`, or uses data specified by the user - for example dummy data that can be included as part of the package. It then uses the other model functions in the R folder to perform different steps of the modelling process, as shown in the schematic below. Those interested in the specific functions can find them in the open-source code at https://github.com/dark-peak-analytics/sick-sickerPack and archived at Mohammed *et al.*[29].

The package contains sample data and parameters, which can be used to run and assess the model. In addition, the function can take a remote path (i.e. a url address), from which remote data can be sourced. It also allows passing credentials, to enable users to access password protected data sources. This can be particularly useful in cases where the model functions are to be separated from (sensitive) parameter information (e.g. prices, survival data), to allow for external - or even public - model review by third parties without the provision of sensitive data. Independently of the 'asserts' within the functions themselves, designed to ensure that functions are not misused, unit tests were written for each function to verify if the function does meet a set defined criteria or rules. Each time the code base is changed the tests can be re-run to ensure that the changes do not inadvertently cause errors or have unintended effects on other parts of the code. As a result, a well thought through and thorough test-bed can give reviewers confidence in the model code. This can reduce the burden of reviewing models since reviewers may only need to ensure that the tests provide adequate coverage, rather than create them themselves.

Discussion

In this paper, we presented the advantages of using R packages for health economic evaluation models and provided a tutorial for creating a basic R package. We also provided a case study where an existing model (the Sick Sicker model) has been converted into a stand-alone R package with documentation and unit tests included. An interested party could install the package from R, run the model with the dummy data, or the sensitive data from the remote server if they have been provided with the key, and see how each function works,

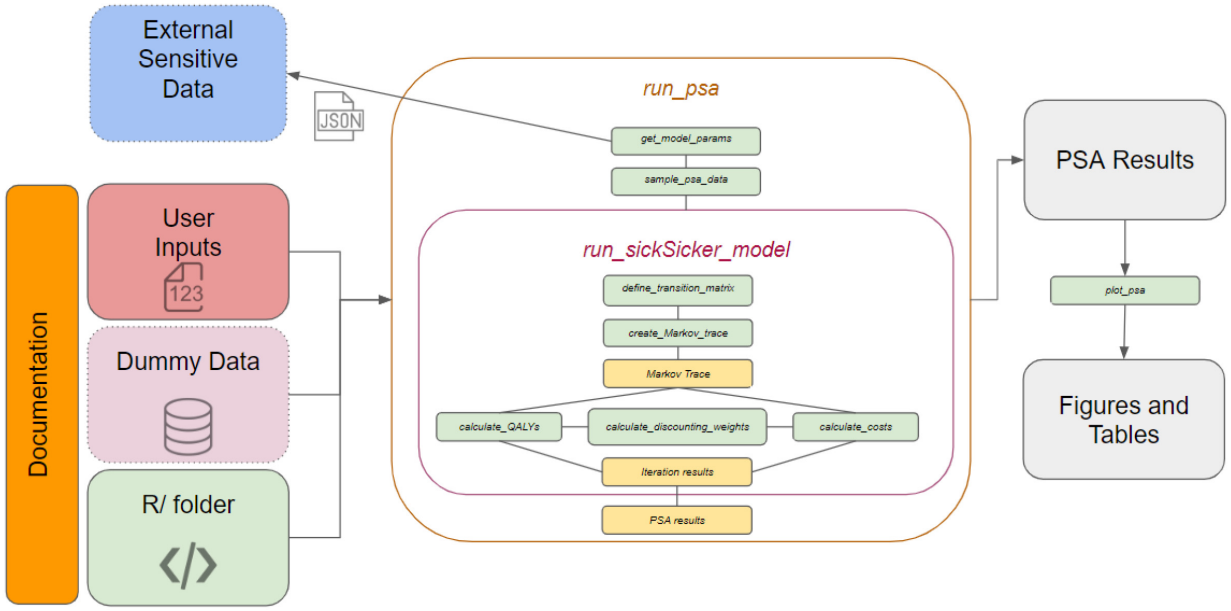

Figure 11. Schematic showing how the sicksickerPack R Package works.

and read a technical report contained in the vignette folder. Throughout, we have argued that more widespread use of R packages would enhance reproducibility, scalability, collaboration, and model validation in health economics and decision science, ultimately benefiting the health economics community as a whole.

The presented decision-analytic models' modularisation and packaging concepts go beyond the framework suggested by Alarid-Escudero *et al.*, 2019[5], by creating functions that are applicable to a specific use-case but thinking about making functions that are generic enough for other model builds, and can therefore be reused in other projects and by other modellers. Furthermore, by including the option to use dummy data contained in the package, as well as data stored on a remote server (if the user has the key/password), the model is truly independent of the data on which it relies, allowing experts in health economic modelling and software development to review the model without necessarily requiring access to sensitive data (and all of the associated administrative burden).

By adopting R packages as a standard approach, health economists can build a large and ever-improving code base, thereby reducing the cost and increasing the quality of health economic evaluation[30]. Building a culture by which model code is shared by default will help overcome the reluctance of some to share their model code, addressing the finding of Emerson *et al.*[6] that many health economists do "not want to confront the issue of publishing their source code, or at the very least, may not view source code publication as a priority". The modular nature of R packages enables efficient updates and integration with other tools, while the comprehensive documentation can aid transparency and facilitate peer review[31]. This has the overall effect of more rapidly establishing and disseminating best practices in health economic evaluation methodologies.

There are several potential challenges to this approach. Firstly, it is important to recognize that the successful adoption of R packages in health economics relies on the trust and confidence of health economists and statisticians in the packages they use. Trusted experts and institutions could play a crucial role in approving or preferring specific packages which may go some way to providing legitimacy. The validation processes, including peer review, certification, and open-source community validation, require ongoing effort and substantial investment but is considered routine in other industries[17].

Open source software is widely accepted and forms the foundation for many crucial software applications including the internet, online banking, automotive, and aerospace software infrastructure. Because it is open to peer review and contribution from the entire world, it tends to become more reliable and secure than commercial software[32]. Applying the same methods to health economic evaluation is likely to improve quality and transparency of decision models. Encouraging the health economics community to contribute to opensource projects, participate in peer reviews, and share their expertise will be crucial in maximising the benefit of open-source software.

However, there is a danger that convergence on the use of specific functions from existing packages will lead to methodological complacency and homogeneity of approach where it is not appropriate. This argument is made in technological advances in all fields and is rarely actually observed. It is much more likely that time freed by having more standardised,

routinely used methods will enable health economists to focus on the specifics of that particular model. Another potential limitation is concern over intellectual property, in both modelling methods and data. While concerns about data can be overcome[15], the benefits to the health economics community from the process described would be much smaller if modelling packages are not made open-source. Finally, and most significantly, the benefits of this approach will not be fully realised unless enough health economists are able to program models in script-based programming languages like R and Python. It is essential that training is provided to both students of health economics, as is becoming commonplace at University MSc programmes, and experts in industry and consulting environments. We hope this paper and our previous works all published open-access go some way to mitigating this constraint.

Future research should focus on exploring avenues for package validation, identifying best practices, and developing teaching tools and resources to facilitate the adoption of R packages in health economics.

Conclusions

This paper has demonstrated the benefits of using R packages in health economic evaluation models, highlighting their potential to improve reproducibility, scalability, collaboration, and model validation. We have provided a tutorial for creating a basic package and a case study of converting an existing model into a package. By embracing R packages as a standard practice, the health economics community can streamline the development process, enhance code quality, and facilitate knowledge sharing.

As the adoption of R becomes more widespread in health economics, it is crucial to ensure that the community maintains high standards in coding practices, documentation, and validation. Fostering a culture of collaboration, knowledge sharing, and continuous improvement will be essential in achieving these goals. Encouraging health economists to contribute to open-source projects, participate in peer reviews, and share their expertise will help to establish and disseminate best practices in health economic evaluation methodologies. By embracing packages and fostering a collaborative culture, the health economics community can ensure the development of robust and reliable models, ultimately informing better healthcare decision-making and resource allocation. Future research should focus on exploring avenues for package validation, identifying best practices, and developing tools and resources to facilitate the adoption of packages in health economics.

Software availability

Source code for the case study sicksickerPack is available: https://github.com/dark-peak-analytics/sicksickerPack

Archived source code for the case study sicksickerPack at time of publication: https://doi.org/10.5281/zenodo.8075587[29]

Source code for the tutorial package HECONpack is available: https://github.com/dark-peak-analytics/HECONpack

Archived source code for the tutorial package HECONpack at time of publication: https://doi.org/10.5281/zenodo.8075580[24]

Acknowledgements

We thank the attendees of the R-HTA conference and colleagues from ScHARR and Dark Peak Analytics for their input.

References

1. Miller DL, Robinson MD, Claxton K, *et al.*: **Health economic modelling in the 21st century: a scoping review of methods applications and challenges.** *Lancet.* 2019; **393**(10187): 2234–2244.
2. Campbell M, Claxton AG, Sculpher MJ: **The use of health economic models in policy making: a systematic review.** *J Health Econ.* 2016; **48**: 1–26.
3. Coyle DA, Claxton AG, Sculpher MJ: **Spreadsheet modelling in health economics: a systematic review of methods and applications.** *Health Econ.* 2020; **29**(1): 1–18.
4. Baio G, Heath A: **When simple becomes complicated: why excel should lose its place at the top table.** 2017; **4**(1).
Publisher Full Text
5. 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): 1329–1339.
PubMed Abstract | Publisher Full Text | Free Full Text
6. Emerson J, Bacon R, Kent A, *et al.*: **Publication of decision model source code: attitudes of health economics authors.** *Pharmacoeconomics.* 2019; **37**(11): 1409–1410.
PubMed Abstract | Publisher Full Text | Free Full Text
7. Wickham H: **R packages: organize, test, document, and share your code.** " O'Reilly Media, Inc.", 2015.
Reference Source
8. Hornik K: **The comprehensive r archive network.** *Wiley Interdiscip Rev Comput Stat.* 2012; **4**(4): 394–398.
Publisher Full Text
9. Mora-Cantallops M, Sánchez-Alonso S, García-Barriocanal E: **A complex network analysis of the comprehensive r archive network (cran) package ecosystem.** *J Syst Softw.* 2020; **170**: 110744.
Publisher Full Text
10. Dunlop WCN, Mason N, Kenworthy J, *et al.*: **Benefits, challenges and potential strategies of open source health economic models.** *Pharmacoeconomics.* 2017; **35**(1): 125–128.
PubMed Abstract | Publisher Full Text | Free Full Text
11. Incerti D, Jansen JP: **hesim: Health economic simulation modeling and decision analysis.** *arXiv preprint arXiv: 2102.09437.* 2021.
Publisher Full Text
12. Filipovic-Pierucci A, Zarca K, Durand-Zaleski I: **Markov models for health economic evaluation modelling in r with the heemod package.** *Value Health.* 2016; **19**(7): A369.
Publisher Full Text

13. Krijkamp EM, Alarid-Escudero F, Enns EA, *et al.*: **Microsimulation modeling for health decision sciences using r a tutorial.** *Med Decis Making.* 2018; **38**(3): 400–422.
PubMed Abstract | Publisher Full Text | Free Full Text

14. 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

15. Smith RA, Schneider PP, 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

16. Brennan A, Chick SE, Davies R: **A taxonomy of model structures for economic evaluation of health technologies.** *Health Econ.* 2006; **15**(12): 1295–1310.
PubMed Abstract | Publisher Full Text

17. Dasbach EJ, Elbasha EH: **Verification of decision-analytic models for health economic evaluations: an overview.** *Pharmacoeconomics.* 2017; **35**(7): 673–683.
PubMed Abstract | Publisher Full Text

18. van Alphen AMIA, van Hof KS, Gravesteijn BY, *et al.*: **Minimising population health loss in times of scarce surgical capacity: a modelling study for surgical procedures performed in nonacademic hospitals.** *BMC Health Serv Res.* 2022; **22**(1): 1456.
PubMed Abstract | Publisher Full Text | Free Full Text

19. Caulley L, Krijkamp E, Doyle MA, *et al.*: **Cost-effectiveness of direct surgery versus preoperative octreotide therapy for growth-hormone secreting pituitary adenomas.** *Pituitary.* 2022; **25**(6): 868–881.
PubMed Abstract | Publisher Full Text | Free Full Text

20. Wickham H, Hester J, Chang W, *et al.*: **Package 'devtools'.** 2022.

21. Wickham H, Bryan J, Barrett M, *et al.*: **usethis: Automate Package and Project Setup.** R package version 2.2.1, 2023.
Reference Source

22. Wickham H, Danenberg P, Eugster M: **roxygen2: in-line documentation for r.** r package version 6.0.1, 2017.
Reference Source

23. Wickham H: **testthat: Get started with testing.** *R J.* 2011; **3**(1): 5–10.
Publisher Full Text

24. Smith R, Schneider P, Mohammed W: **Heconpack.** 2023.
http://www.doi.org/10.5281/zenodo.8075580

25. Wickham H: **assertthat: Easy Pre and Post Assertions.** R package version 0.2.1, 2019.
Reference Source

26. Wickham H, Danenberg P, Csárdi G, *et al.*: **roxygen2: In-Line Documentation for R.** R package version 7.2.3, 2022.
Reference Source

27. St Laurent AM: **Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software.** " O'Reilly Media, Inc.", 2004.
Reference Source

28. Alarid-Escudero F, Krijkamp E, Enns EA, *et al.*: **An introductory tutorial on cohort state-transition models in r using a cost-effectiveness analysis example.** *Med Decis Making.* 2023; **43**(1): 3–20.
PubMed Abstract | Publisher Full Text | Free Full Text

29. Mohammed W, Smith R, Schneider P: **sick-sickerpack.** 2023.
http://www.doi.org/10.5281/zenodo.8075587

30. Hatswell AJ, Chandler F: **Sharing is caring: the case for company-level collaboration in pharmacoeconomic modelling.** *Pharmacoeconomics.* 2017; **35**(8): 755–757.
PubMed Abstract | Publisher Full Text

31. Sampson CJ, Arnold R, Bryan S, *et al.*: **Transparency in decision modelling: what, why, who and how?** *Pharmacoeconomics.* 2019; **37**(11): 1355–1369.
PubMed Abstract | Publisher Full Text | Free Full Text

32. Hoepman JH, Jacobs B: **Increased security through open source.** *Communications of the ACM.* 2007; **50**(1): 79–83.
Publisher Full Text

Open Peer Review

Current Peer Review Status: ? ?

Version 1

Reviewer Report 24 October 2023

https://doi.org/10.21956/wellcomeopenres.21773.r67576

?

Joe Moss
York Health Economics Consortium, York, UK

Smith *et al.* have written a well-structured paper outlining the benefits of using R packages when constructing health economic models. I agree with the underlying principle that open-source coding would be invaluable to the industry. However, I think a few minor points need to be addressed.

1. The 2nd and 3rd paragraphs of the package build tutorial feel very repetitive and I am wondering if they could be merged in some way

2. I believe the HECONpack should be linked (URL) in the main manuscript just like sicksickerPack was when first introduced (instead of being linked at the end of the manuscript). This would allow the reader to look at the package whilst following along with the steps outlined. This may aid in the reproducibility of the methods.

3. Whilst I agree that open-source coding should be what people strive for, the conclusions of the paper suggest that it is down to the individual economists/statisticians who are holding back. This may be the case in academia but it should be acknowledged that industrial sponsors of economic projects own the intellectual property rights for any R code produced in a project they fund. Therefore, not only will it require a shift in thinking at the individual level it will also require a shift in thinking at an industrial level, but large companies are unlikely to be willing to share code (for which they have paid for) for which they will share publicly for their competitors to use and adapt (for little or no costs). I think it would be good to also acknowledge this point as a limitation in the discussion.

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?
Partly

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
No source data required

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: Statistics; R; R shiny; Health Economics

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.

Reviewer Report 09 October 2023

https://doi.org/10.21956/wellcomeopenres.21773.r67574

?

Isaac Corro Ramos
Erasmus University Rotterdam, Rotterdam, The Netherlands

The authors discuss the benefits of presenting health economic models as R packages. Such benefits include improving reproducibility, collaboration, and model transparency and validation. A tutorial for creating a basic package and a case study of an existing model converted into an R package is provided. The paper is relevant for health economic modelers using R for that purpose.

Specific comments are provided below:

The manuscript could become more concise by deleting some parts that do not carry fundamental information. A few suggestions for deleting text are given below:

1. Page 6, paragraph “To summarise [...] this gap.”. Can be deleted since it is a repetition of text presented in previous parts of the paper.
2. Page 6-8, paragraphs “In this tutorial [...] CRAN or GitHub.”. These paragraphs could be deleted so that “Tutorial: The basics of R Packages for Health Economic Evaluation” starts at “This tutorial demonstrates how to create an R package named”.
3. Page 13, sentences “R functions [...] about the package.” Could be deleted too.

- Figures:

1. The resolution of the figures seems to be low in general. Please consider using high-resolution figures where appropriate.
2. Figure 1 does not represent what the text describes. An algorithm is often represented in pseudo-code, while Figure 1 shows a graphical representation of a "conceptual" model.
3. Page 4 says “Figure 3 above” but it is shown below. Please note there is no need to mention above/below for figures/tables if they are properly referenced. If the authors prefer to do so, please check the “above/below”.
4. Figures 8 and 9 might be deleted since they are not very informative.

Please correct typographical errors throughout the paper since there are a few (all minor). In addition:

1. Please consider using a different font type for R functions, packages, etc.
2. As the authors know, R is case-sensitive, so please carefully check the names provided, e.g.: create_Package

- Abstract: first sentence: I would argue that “increasing model development efficiency…” depends on (possibly many) other factors, not only using R.

- Page 3:

1. Introduction first paragraph: where authors say that R is becoming more popular, it would be nice to cite some R models published. For the pharma industry, for example, I don't think this is still the case, since, in my experience, the majority of models are still developed in spreadsheet software.
2. Sentences “Without a sound […] standards.” I would argue that the text provided applies to spreadsheet models too.
3. Other than having packages on CRAN, please explain what is the advantage compared to for example uploading all relevant files to GitHub as an R project (but not as a package).
4. Please explain how the authors adapted the ‘Sick Sicker’ model, to what extent the structure is different and what are similar principles.
5. Please explain what is meant by “Except for the simplest models, which are typically not programmed in R”.

- Page 4: Sentences “However […] or tables”. Please explain if this is what the authors are proposing or if this is what the Alarid-Escudero paper is proposing.

- Page 6: While I acknowledge the potential benefits of ERGs, in more than 10 years of experience as an ERG member, I have seen just one model submitted by companies in R, all other models were developed in spreadsheet software. Therefore, the impact on STA’s might still be very limited. ERG groups on the other hand may be more inclined to program in R. We have done it for example for NICE Diagnostic Assessments (see e.g., SeHCAT). DAPs are reviewed after a few years, and sometimes by other ERGs, so here I see a more immediate benefit. I’d like the authors to reflect on this if they consider it appropriate.

- Page 12: Paragraph “In the long run […] review them”. I believe this is a very important message. It may deserve a more prominent place in the paper, maybe at the end of the conclusions/recommendations. Consider also making this a stronger point since the majority of models being assessed by ERGs lack any technical documentation. This is one of the recommendations in guidelines such as ISPOR TF on model transparency but in practice, it seems that HE modelers have problems adhering to it.

- Comments on the tutorial section:
- Page 8: At command DevTools:: create_package(path = "HECONpack"). I got an Error: 'create_package' is not an exported object from 'namespace:devtools'. The code worked when DevTools:: was removed. Please consider making readers aware of this. Less experienced users might not try further and leave the tutorial here.
- Page 8: when referring to “the” path, maybe for less experienced users the authors might want to explain getwd() and setwd() commands.
- Page 8: Table 1 - I also obtained a .gitignore txt document.
- Page 8: the links in the sentence “More information on R projects can be found here and for more guidance on package setup more generally please refer to this link” are both the same and equal to the one previously mentioned. Please check.
- Page 9: The assertthat::assert_that() checks that all of the arguments are class numeric. Technically this is true because .numeric() is used afterwards. Please check.
- Page 9: “For example help("calcICER") generates the help file shown in Figure 8 below”. Following the steps in the tutorial, this step did not work: help("calcICER") returns No documentation for ‘calcICER’ in specified packages and libraries: you could try ‘??calcICER’
- Page 11: Please clarify whether the DESCRIPTION file is a text file that can be changed manually.
- Page 11, box with test_that code: The code is missing }) at the end.
- Page 11: “The following command runs all tests within the package’s “tests/testthat” folder:”. The box with the code is empty.
- Page 11: “Alternatively, the RStudio keyboard shortcut Ctrl + Shift + T on a Windows machine or Cmd + Shift + T on a Mac can be used to run the tests.”. There is also a button on RStudio called Run Tests.
- Page 12, box with df_res_example code: It might be good to clarify if this code needs to be written in the RStudio console, in an R script, or whether it does not matter.
- Page 12: “When users build HECONpack it will be included in the final package distribution”. It might be good to clarify that this seems to work only after the package is created. If I "Knit" the code now one would get an error.
- Page 12: “This then enables the user to call the library("HECONpack") to load the package functions and data as they would with any other installed package.” Following the steps in the tutorial, the package seems to be loaded but R cannot find the function calcICER.
- Comments on the case study section:

- Page 13: “A user of the model would need to run…”. Before this point, I would expect the authors to explain how can one download the code of the package from Git Hub, install it on one's computer, and get it ready to use. Please consider adding this explanation to guide readers through the process. In the same paragraph, I would suggest adding specific numerical examples, either here or in an appendix.

- Page 13: the paragraph starting “The package contains…” I would suggest giving examples of how to do all the steps described. It is also unclear what “parameters” and “function” the authors mean here. Please clarify.

- Page 13: “Each time the code base is changed the tests can be re-run”. Can or should?

- Page 13: Please revise the sentence “As a result, a well thought through and thorough test-bed can give reviewers confidence in the model code.”

- Comments on the Discussion section:

- Page 13: “We also provided a case study where an existing model (the Sick Sicker model) has been converted into a stand-alone R package with documentation and unit tests included.” I think the authors have not shown this. They have shown the final "product" on Git Hub, but they do not guide how this package was created. Showing this process could be useful for readers if, for example, they want to create a new version of the package.

- Page 14: “Furthermore […] burden”. This is important because sharing sensitive data is seen as one of the barriers to the adoption of open-source models. As mentioned in a previous comment, the authors could provide an example, so that readers can replicate and understand how this process works. Likewise, in the conclusions “and a case study of converting an existing model into a package”; I don’t think the authors have shown this.

- Page 14: “Trusted experts and institutions could play a crucial role in approving or preferring specific packages which may go some way to providing legitimacy.”. The authors might refer here to the Dutch PharmacoEconomic guidelines developed by the Dutch Healthcare Institute (ZIN) in which R is accepted as a valid software to program HE models, but only some packages are accepted.
https://english.zorginstituutnederland.nl/publications/publications/2022/12/15/guideline-for-building-cost-effectiveness-models-in-r

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?

Partly

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: Health econmics researcher with more than 10 years experience in modelling and more than 15 years experience with R.

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.