Living HTA

Automating HTA with R

Monday 23rd May 2022 | R-HTA Workshop, Oxford UK

Lumanity

Robert Smith[1,2,3], Paul Schneider[2,3] & Wael Mohammed[2,3]

1 Lumanity, Sheffield

2 Dark Peak Analytics, Sheffield

3 ScHARR, University of Sheffield

Living HTA: Automating HTA with R

Robert Smith [1,2,3] & Paul Schneider [2,3] & Wael Mohammed [2,3]

[1] Lumanity, Sheffield, [2] ScHARR, University of Sheffield, [3] Dark Peak Analytics, Sheffield

Background The process of updating economic models is time-consuming and expensive, and often involves the transfer of sensitive data between parties. Here, we demonstrate how HEOR can be conducted in a way that allows companies to retain full control of their data, while automating reporting as new information becomes available.

Method We developed an automated analysis and reporting pipeline for health economic modelling and made the source code openly available on a GitHub repository. It consists of three parts:

- An economic model is constructed by the consultant using pseudo data (i.e. random data, which has the same format as the real data).
- On the company side, an application programming interface (API), generated using the R package plumber, is hosted on a server. An automated workflow is created. This workflow sends the economic model to the company API. The model is then run within the company server. The results are sent back to the consultant, and a (PDF) report is automatically generated using RMarkdown.
- This API hosts all sensitive data, so that data does not have to be provided to the consultant.

Results & Discussion The method is relatively complex, and requires a strong understanding of R, APIs, RMarkdown and GitHub Actions. However, the result is a process, which allows the consultant to conduct health economic (or any other) analyses on company data, without having direct access – the company does not need to share their sensitive data. The workflow can be scheduled to run at defined time points (e.g. monthly), or when triggered by an event (e.g. an update to the underlying data or model code). Results are generated automatically and wrapped into a full report. Documents no longer need to be revised manually.

Conclusions This example demonstrates that it is possible, within a HEOR setting, to separate the health economic model from the data, and automate the main steps of the analysis pipeline. We believe this is the first application of this procedure for a HEOR project.

Agenda

1.

Background

What problems exist?

Lumanity

Background

There are several changes occurring simultaneously in health economics & decision science.

1) **New technologies - shift to open-source packages (e.g. Rmarkdown, Shiny, BCEA, Heemod).**

   Previous presentations on Shiny to make R more accessible, but why R? Why not Python. APIs help mitigate this.

2) **Big data - Increased use of large datasets, and concerns around data-security.**

   We will show a method of allowing companies to retain control of their data.

3) **Big simulation – incorporating uncertainty in health economic models.**

   By running models on remote servers, we can easily scale up computing power.

4) **Shift from recommendations from 'static' to 'dynamic' (Living)**

   "The ability to link real-world evidence with evidence-based practice will drive a shift from recommendations being produced at a single 'static' point in time to more dynamic, living guidance, and from health technology assessment to health technology management." NICE strategy 2021 to 2026: Dynamic, Collaborative, Excellent

Background … continued

Wellcome Open Research

Wellcome Open Research 2020, 5:69 Last updated: 23 MAR 2022

Check for updates

METHOD ARTICLE

REVISED **Making health economic models Shiny: A tutorial**

[version 2; peer review: 2 approved]

Robert Smith*, Paul Schneider*

School of Health and Related Research, University of Sheffield, Regents Court, Sheffield, S1 4DA, UK

* Equal contributors

v2 **First published:** 14 Apr 2020, **5**:69
https://doi.org/10.12688/wellcomeopenres.15807.1
Latest published: 31 Jul 2020, **5**:69
https://doi.org/10.12688/wellcomeopenres.15807.2

Abstract

Health economic evaluation models have traditionally been built in Microsoft Excel, but more sophisticated tools are increasingly being used as model complexity and computational requirements increase. Of all the programming languages, R is most popular amongst health economists because it has a plethora of user created packages and is highly flexible. However, even with an integrated development environment such as R Studio, R lacks a simple point and click user interface and therefore requires some programming ability. This might make the switch from Microsoft Excel to R seem daunting, and it might make it difficult to directly communicate results with decisions makers and other stakeholders.

The R package Shiny has the potential to resolve this limitation. It allows programmers to embed health economic models developed in R into interactive web browser based user interfaces. Users can specify their own assumptions about model parameters and run different scenario analyses, which, in the case of regular a Markov model, can be computed within seconds. This paper provides a tutorial on how to wrap a health economic model built in R into a Shiny application. We use a four-state Markov model developed by the Decision Analysis in R for Technologies in Health (DARTH) group as a case-study to demonstrate main principles and basic functionality.

A more extensive tutorial, all code, and data are provided in a GitHub repository.

Keywords

Health Economics, R, RShiny, Decision Science

Open Peer Review

Approval Status ✓ ✓

	1	2
version 2 (revision) 31 Jul 2020	✓ view	✓ view
version 1 14 Apr 2020	? view	✓ view

1. **Talitha L. Feenstra**, University of Groningen, Groningen, The Netherlands

2. **Yiqiao Xin**, University of Glasgow, Glasgow, UK

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

- Our previous **tutorial paper** showed how to take a model in R and host it in an **R-Shiny** application.
- This helps to **make models in R more usable by humans**.
- However, this often **requires data to be shared with a consultant**.
- Also **requires a lot of human interaction**, for example would need to be updated if data is updated.
- Our focus now turns to **developing a framework where data doesn't need to be shared**, and **model reports/applications can be 'living' to reflect the latest data**.

Lumanity

2.

Concept

How to implement this …

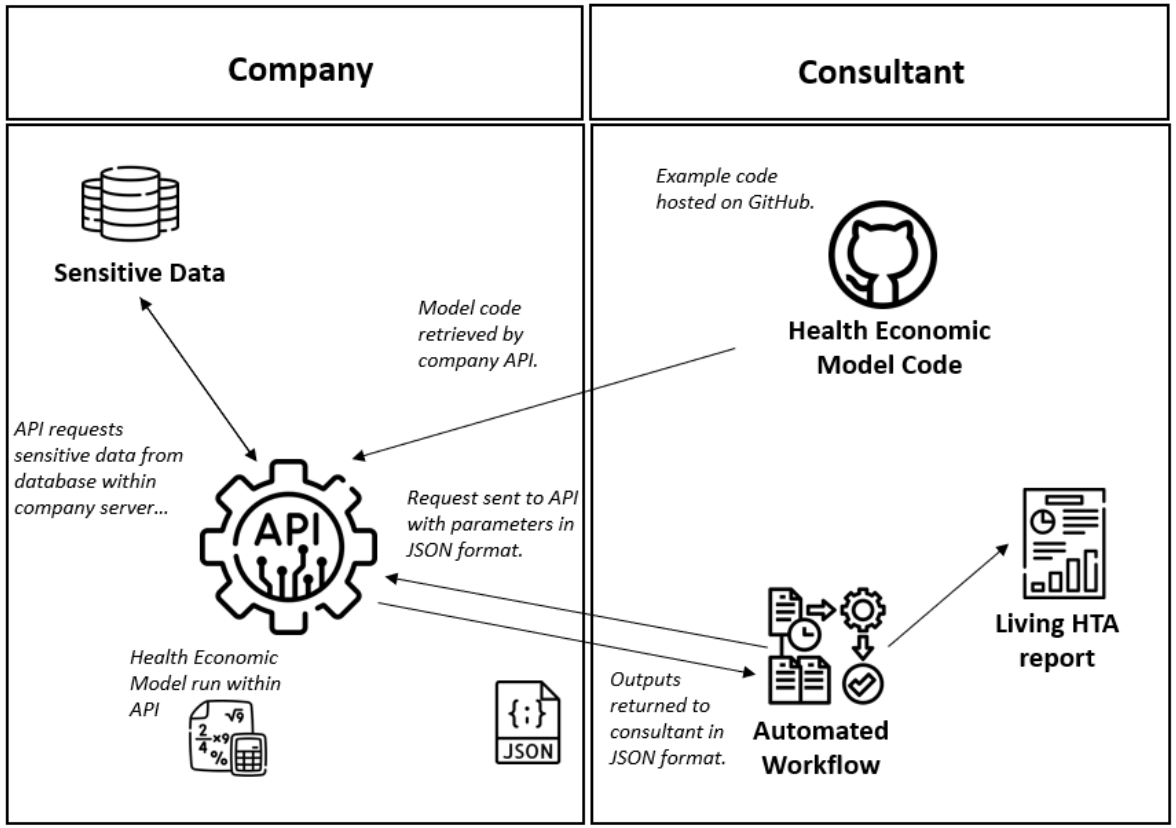
Lumanity

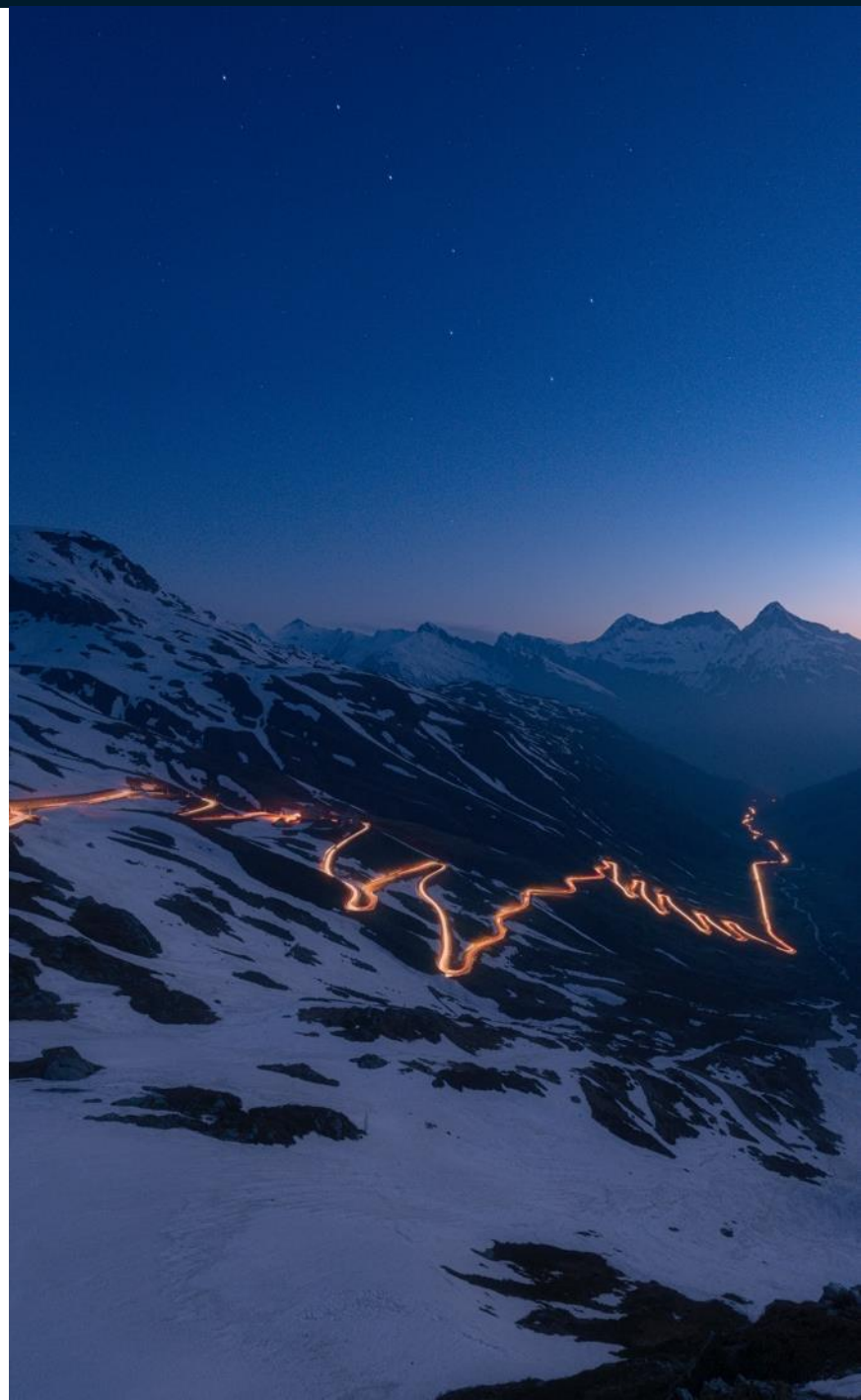
What we will show today …

Using open-source software to generate an automated HTA workflow…

- Construct a script-based health economic models in R. Although could be Python, Ruby, Julia ….
- Host health economic model on GitHub repository, provide access to company.
- Host API on company server. The API will:
  - … have access to sensitive data.
  - … enable users to run R code on it.
  - … allow users to pass R code to it.
  - … return model results from it.
- Write script to pass model inputs & code to company API & generate a PDF report using markdown.
- Create a workflow which runs the script on schedule, or as new evidence emerges (e.g. when a data-base is updated).
- Contain all this functionality within a Shiny app – to enable non-programmers to interact with the API.

Living HEOR: Automating HTA with R





Lumanity

3.

Hosting a HE model in an API

Creating an *Application Programming Interface* to host a health economic model using the plumber R package.

Lumanity

Hosting a HE model in an API

```r
################

library(dampack)
library(readr)
library(assertthat)

#* @apiTitle Client API hosting sensitive data
#*
#* @apiDescription This API contains sensitive data, the client does not
#* want to share this data but does want a consultant to build a health
#* economic model using it, and wants that consultant to be able to run
#* the model for various inputs
#* (while holding certain inputs fixed and leaving them unknown).

#* Run the DARTH model
#* @serializer csv
#* @param path_to_psa_inputs is the path of the csv
#* @param model_functions gives the github repo to source the model code
#* @param param_updates gives the parameter updates to be run
#* @post /runDARTHmodel
function(path_to_psa_inputs = "parameter_distributions.csv",
         model_functions = paste0("https://raw.githubusercontent.com/",
                                  "BresMed/plumberHE/main/R/darth_funcs.R"),
         param_updates = data.frame(
           parameter = c("p_HS1", "p_S1H"),
           distribution = c("beta", "beta"),
           v1 = c(25, 50),
           v2 = c(150, 70)
         )) {


  # source the model functions from the shared GitHub repo...
  source(model_functions)

  # read in the csv containing parameter inputs
  psa_inputs <- as.data.frame(readr::read_csv(path_to_psa_inputs))

  # for each row of the data-frame containing the variables to be changed...
  for(n in 1:nrow(param_updates)){

  # update parameters from API input
  psa_inputs <- overwrite_parameter_value(
                           existing_df = psa_inputs,
                           parameter = param_updates[n,"parameter"],
                           distribution = param_updates[n,"distribution"],
                           v1 = param_updates[n,"v1"],
                           v2 = param_updates[n,"v2"])
  }

  # run the model using the single run-model function.
  results <- run_model(psa_inputs)

  # check that the model results being returned are the correct dimensions
  # here we expect a single dataframe with 6 columns and 1000 rows
  assertthat::assert_that(
    all(dim(x = results) == c(1000, 6)),
    class(results) == "data.frame",
    msg = "Dimensions or type of data are incorrect,
  please check the model code is correct or contact an administrator.
  This has been logged"
  )

  # check that no data matching the sensitive csv data is included in the output
  # searches through the results data-frame for any of the parameter names,
  # if any exist they will flag a TRUE, therefore we assert that all = F
  assertthat::assert_that(all(psa_inputs[, 1] %in%
        as.character(unlist(x = results,
                            recursive = T)) == F))

  return(results)
}
```

- Load necessary packages
- Describe the API, used in Swagger
- Roxygen style documentation for function, what are the inputs…
- Source the model functions from GitHub
- Overwrite default data with non-sensitive inputs
- Run the model
- Check the results object doesn't contain sensitive data
- Return the results object

Lumanity

4.

Calling the API from R

Passing parameters and code to the health economic model & returning results…

Lumanity

Running the model – calling the API

```r
# remove all existing data from the environment.
rm(list = ls())

library(ggplot2)
library(jsonlite)
library(httr)

# run the model using the connect server API
results <- httr::content(
  httr::POST(
    # the Server URL can also be kept confidential, but will leave here for now
    url = "https://connect.bresmed.com",
    # path for the API within the server URL
    path = "rhta2022/runDARTHmodel",
    # code is passed to the client API from GitHub.
    query = list(model_functions =
                   paste0("https://raw.githubusercontent.com/",
                          "BresMed/plumberHE/main/R/darth_funcs.R")),
    # set of parameters to be changed ...
    # we are allowed to change these but not some others
    body = list(
      param_updates = jsonlite::toJSON(
        data.frame(parameter = c("p_HS1","p_S1H"),
                   distribution = c("beta","beta"),
                   v1 = c(25, 50),
                   v2 = c(150, 100))
      )
    ),
    # we include a key here to access the API ... like a password protection
    config = httr::add_headers(Authorization = paste0("Key ",
                                                       Sys.getenv("CONNECT_KEY")))
  )
)

# write the results as a csv to the outputs folder...
write.csv(x = results,
          file = "outputs/darth_model_results.csv")

source("report/makeCEAC.R")
source("report/makeCEPlane.R")

# render the markdown document from the report folder,
# passing the results dataframe to the report.
rmarkdown::render(input = "report/darthreport.Rmd",
                  params = list("df_results" = results),
                  output_dir = "outputs")
```

Load necessary packages

Call the API:

- URL and path combine to identify where the API is.
- Query and body both allow for inputs to be provided to the API… We convert the data-frame of inputs to JSON first.
- Config allows us to add the KEY – which is hidden as an environment variable.
- **Result of the API is stored as an object** (results).

Write the results to a csv… not strictly necessary.

Render an Rmarkdown document based on the results of the API call, store the document in ‘outputs’ directory.

5.

Automating health economic model updates

Automating health economic model updates using GitHub Actions and R-markdown.

Lumanity

Automating health economic model updates

```yaml
on:
  push:
    branches:
    - main
  schedule:
    - cron: '1 1 1 * *'

name: Run DARTH model on client API
jobs:
  createPullRequest:
    runs-on: windows-2019
    env:
      GITHUB_PAT: ${{ secrets.GITHUB_TOKEN }}
  # Load repo and install R
    steps:
    - uses: actions/checkout@master
    - uses: r-lib/actions/setup-r@master

    - name: Setup pandoc
      uses: r-lib/actions/setup-pandoc@v2
      with:
        pandoc-version: '2.17.1.1'

    - name: Install TinyTeX
      uses: r-lib/actions/setup-tinytex@v2
      env:
          # install full prebuilt version
          TINYTEX_INSTALLER: TinyTeX

    - name: Install dependencies
      run: |
          install.packages(
          c("reshape2","jsonlite","httr", "readr", "rmarkdown", "markdown")
          )
          install.packages(
          "scales", dependencies = TRUE, repos = 'http://cran.rstudio.com/'
          )
          install.packages(
          "ggplot2", dependencies = TRUE, repos = 'http://cran.rstudio.com/'
          )
      shell: Rscript {0}

    - name: Run the model from API and create report
      env:
         CONNECT_KEY: ${{secrets.PLUMBER_SECRET}}
      run: |
          source("scripts/run_darthAPI.R")
      shell: Rscript {0}

    - name: Create Pull Request
      uses: peter-evans/create-pull-request@v3
      with:
        token: ${{ secrets.GITHUB_TOKEN }}
        commit-message: Automated Model Run from API
        title: 'Living HTA Automated Model Run'
        body: >
          Automated model run
        labels: report, automated pr
```

Schedule jobs based upon a **push to the main branch, or at a scheduled time** (00:01 on 1st of the month).

Set-up code:

- Start running a Windows 2019 server.
- Checkout the repository.
- Set up R.
- Install all dependencies

Run the script querying the API and creating a report with Rmarkdown.

Create a pull request to the repository with the new results csv and markdown report included.

6.

Discussion

What are the implications of these methods…

Lumanity

What are the pros and cons of this framework.

Advantages of this framework

- **Security -** Data owners retain control of their data. No data need leave the data-owner's servers.
- **Transparency -** Separating the model code from the data can significantly improve the transparency of the health economic model. Many models could be passed to the data, not just one!
- **Computational Power** - The computational burden of the model is handled on a remote server.
- **Storage** – Larger datasets can be analyzed than would be possible on a laptop.
- **Living analysis** - API calls can be made at any time. A decision maker can see a report that will always reflect the data held by the company.

Disadvantages of this framework

- **Security -** Likely to remain concerns about data security, even with the authentication procedures built into the API functionality.
- **Transparency -** Risk that running the model remotely will result in the perception that the model is a 'black box' (I'd disagree!).
- **Coding practice -** The model code needs to be versatile enough to manage unknown data updates. *Proper testing will help mitigate these risks.*
- **Technical skillset -** This is not commonly implemented, or a common skill-set among health economists. Most models are not built in R.

1) **What are the needs of different stakeholders? The company, the consultant, the HTA body, the ERG?**
2) **When is this likely to be most useful?**
3) **Is this too big a leap – can we pull HEOR this far?**

Thank you

More information about this presentation can be found at:

Open-source code: RobertASmithBresMed/plumberHE: Health Economics using Plumber APIs (github.com)
Draft paper: plumberHE/academicpaper.pdf at main · RobertASmithBresMed/plumberHE (github.com)

More information about the author's organizations can be found at:

Lumanity
Dark Peak Analytics
ScHARR, University of Sheffield

Lumanity