Python for Health Technology Assessment: Why R Shouldn’t be the Only Language at the Table

DEVIN INCERTI

JUNE 12, 2023

entityrisk

Why not just R?

- ”R for HTA” should (in my opinion) be about **encouraging good coding practices** and use of **fit for purpose tools**, not switching from one dominant tool (Excel) to another (R)
- All languages have strengths and weaknesses; they can **complement** (rather than substitute for) one another:
  - interpreted languages: Python, Julia, R
  - compiled languages: C, C++, Fortran
  - probabilistic programming: Stan, JAGS, BUGS

Why Python?

- Large and highly collaborative developer and user community
  - Always improving
  - State-of-the art tooling
  - General purpose
  - Dominant language in machine learning (TensorFlow, PyTorch, scikit-learn)
- Excellent scientific libraries:
  - pandas, numpy, scipy, xarray
  - Reduces need for using compiled code like C++ (see slide 10)
- Classes are first class citizens
- Simple syntax and an official style guide (PEP 8) makes both learning and automating styling easy
- Type checking (type hints, mypy)

Language	Share	Trend
Python	27.66 %	+0.2 %
Java	16.16 %	-1.3 %
JavaScript	9.44 %	-0.2 %
C#	6.79 %	-0.5 %
C/C++	6.6 %	-0.1 %
PHP	5.04 %	-0.4 %
R	4.17 %	-0.3 %

Most "popular" programming languages according to PYPL, which bases rankings on Google searches for language tutorials

Importing packages

```python
import numpy as np
import xarray as xr
```

```r
library(tidyverse)
```

Constructing transition probabilities

```python
def trans_prob_matrix():
    """Create an example 3x3 transition probability matrix.

    Returns
    -------
    numpy.ndarray
        An n_states by n_states array where n_states is the
        number of health states.
    """
    return np.array(
        [
            [0.8, 0.1, 0.1],
            [0.0, 0.5, 0.5],
            [0.0, 0.0, 1.0]
        ]
    )
```

```r
#' Create an example 3x3 transition probability matrix.
#'
#' @return An n_states by n_states matrix where n_states is the
#' number of health states.
trans_prob_matrix <- function() {
  matrix(
    c(
      0.8, 0.1, 0.1,
      0.0, 0.5, 0.5,
      0.0, 0.0, 1.0
    ),
    byrow = TRUE,
    nrow = 3
  )
}
```

Simulating a simple Markov model

```python
def sim_markov(trans_probs, n_cycles=5):
    """Simulate disease progression with a Markov model.

    Parameters
    ----------
    trans_probs : numpy.ndarray
        A 2D numpy array containing probabilities of transition
        between health states.
    n_cycles : int, default=5
        The number of cycles to simulate the model for.
        Default is 5.

    Returns
    -------
    numpy.ndarray
        An n_cycles + 1 by n_states array storing state
        occupancy probabilities by model cycle.
    """
    state_probs = np.empty((n_cycles + 1, 3))  # The Markov trace
    state_probs[0, :] = [1, 0, 0]  # Everyone starts in the same state
    for t in range(n_cycles):  # Python indexing starts at 0
        state_probs[t + 1, :] = state_probs[t, :] @ trans_probs
    return state_probs
```

```r
#' Simulate disease progression with a Markov model.
#'
#' @param trans_probs A matrix containing probabilities of transitions
#' between health states.
#' @param n_cycles The number of cycles to simulate the model for.
#' Default is 5.
#'
#' @return A n_cycles + 1 by n_states matrix storing state occupancy
#' probabilities by model cycle.
sim_markov <- function(trans_probs, n_cycles = 5) {
  state_probs <- matrix(NA, nrow = n_cycles + 1, ncol = 3) # The Markov trace
  state_probs[1, ] <- c(1, 0, 0) # Everyone starts in the same state
  for (t in 1:n_cycles) { # R indexing starts at 1
    state_probs[t + 1, ] <- state_probs[t, ] %*% trans_probs
  }
  state_probs
}
```

Simulating a simple Markov model: output

```
>>> import functions as hta
>>> trans_probs = hta.trans_prob_matrix()
>>> state_probs = hta.sim_markov(trans_probs, n_cycles=5)
>>> print(trans_probs)
[[0.8 0.1 0.1]
 [0.  0.5 0.5]
 [0.  0.  1. ]]
>>> print(state_probs)
[[1.      0.      0.     ]
 [0.8     0.1     0.1    ]
 [0.64    0.13    0.23   ]
 [0.512   0.129   0.359  ]
 [0.4096  0.1157  0.4747 ]
 [0.32768 0.09881 0.57351]]
```

```
> source("functions.R")
> trans_probs <- trans_prob_matrix()
> state_probs <- sim_markov(trans_probs, n_cycles = 5)
> print(trans_probs)
     [,1] [,2] [,3]
[1,]  0.8  0.1  0.1
[2,]  0.0  0.5  0.5
[3,]  0.0  0.0  1.0
> print(state_probs)
        [,1]    [,2]    [,3]
[1,] 1.00000 0.00000 0.00000
[2,] 0.80000 0.10000 0.10000
[3,] 0.64000 0.13000 0.23000
[4,] 0.51200 0.12900 0.35900
[5,] 0.40960 0.11570 0.47470
[6,] 0.32768 0.09881 0.57351
```

Computing QALYs

```python
def compute_qalys(state_probs, qol, discount_rate=0.03):
    """Compute discounted quality-adjusted life-years (QALYs).

    Parameters
    ----------
    state_probs : numpy.ndarray
        An n_cycles + 1 by n_states array containing state occupancy
        probabilities.
    qol : list or numpy.ndarray
        A 1D array_like of length n_states containing quality-of-life
        (QoL) weights for each health state.
    discount_rate : float, default=0.03
        Discount rate for QALYs.
    """
    qalys = state_probs @ qol
    n_years = state_probs.shape[0]  # Assume each cycle is a year
    times = np.arange(0, n_years, step=1)  # Range is from [state, stop)
    discounted_qalys = qalys / (1 + discount_rate) ** times
    return discounted_qalys
```

```r
#' Compute discounted quality-adjusted life-years (QALYs).
#'
#' @param state_probs An n_cycles + 1 by n_states matrix containing
#' state occupancy probabilities.
#' @param qol A length n_states vector containing quality-of-life
#' (QoL) weights for each health state.
#' @param discount_rate Discount rate for QALYs.
compute_qalys <- function(state_probs, qol, discount_rate = 0.03) {
  qalys <- state_probs %*% qol
  n_years <- nrow(state_probs) # Assume each cycle is a year
  times <- seq(0, n_years - 1, by = 1) # Range is from [state, stop]
  discounted_qalys <- qalys / (1 + discount_rate)^times
  discounted_qalys
}
```

Computing QALYs: output

```
>>> discounted_qalys = hta.compute_qalys(
...     state_probs, qol=[0.8, 0.6, 0]
... )
>>>
>>> print(discounted_qalys)
[0.8        0.67961165 0.55613159 0.44567399 0.35281821 0.27726834]
```

```
> discounted_qalys <- compute_qalys(
+     state_probs, qol = c(0.8, 0.6, 0)
+ )
> print(discounted_qalys)
          [,1]
[1,] 0.8000000
[2,] 0.6796117
[3,] 0.5561316
[4,] 0.4456740
[5,] 0.3528182
[6,] 0.2772683
```

Python advantages specific to decision modeling

- N-dimensional matrix multiplication makes vectorizing Markov models easy without needing compiled languages like C++
  - Probabilistic sensitivity analysis (PSA)
  - Multiple treatment strategies
  - Multiple cohorts or subgroups
- Labeled arrays make operations with high dimensional objects simpler
  - Parameter simulations, treatment strategies, individuals/subgroups, time periods, …

Vectorizing the simulation of the Markov model while performing a PSA

```python
def sim_markov_psa(trans_probs, n_cycles=5):
    """Simulate disease progression with a Markov model and use probabilistic
    sensitivity analysis (PSA) to propagate parameter uncertainty.

    Parameters
    ----------
    trans_probs : numpy.ndarray
        An n_sims by n_states by n_states 3D array where each slice is a
        transition probability matrix for a given draw from the PSA.
    n_cycles : int, default=5
        The number of cycles to simulate the model for. Default is 5.

    Returns
    -------
    numpy.ndarray
        An n_sims by n_cycles + 1 by n_states array storing state occupancy
        probabilities by model cycle.
    """
    n_sims = trans_probs.shape[0]  # First axis is the simulation axis
    state_probs = np.empty((n_sims, n_cycles + 1, 3))  # The Markov trace
    state_probs[:, 0, :] = [1, 0, 0]  # Everyone starts in the same state
    for t in range(n_cycles):  # Python indexing starts at 0
        state_probs[:, [t + 1], :] = state_probs[:, [t], :] @ trans_probs
    return state_probs
```

This simply simulates a 3D array of transition probability matrices with Dirichlet distributions

```python
>>> import functions as hta
>>> trans_probs = hta.trans_prob_matrix()
>>> SAMPLE_SIZE = 100
>>> trans_data = (trans_probs * SAMPLE_SIZE).astype(int)
>>> trans_probs_psa = hta.simulate_trans_probs(trans_data, n_sims=3)
>>> state_probs_psa = hta.sim_markov_psa(trans_probs_psa, n_cycles=5)
>>> print(state_probs_psa)
[[[1.         0.         0.        ]
  [0.74729753 0.         0.        ]
  [0.5584536  0.         0.        ]
  [0.41733099 0.         0.        ]
  [0.31187042 0.         0.        ]
  [0.23305999 0.         0.        ]]

 [[1.         0.         0.        ]
  [0.81609223 0.         0.        ]
  [0.66600653 0.         0.        ]
  [0.54352276 0.         0.        ]
  [0.4435647  0.         0.        ]
  [0.36198971 0.         0.        ]]

 [[1.         0.         0.        ]
  [0.81540843 0.         0.        ]
  [0.66489091 0.         0.        ]
  [0.54215765 0.         0.        ]
  [0.44207992 0.         0.        ]
  [0.36047569 0.         0.        ]]]
```

Labeling high-dimensional arrays simplifies code and understanding considerably, while still preserving computational efficiency

```python
def label_state_probs(state_probs):
    """Convert state occupancy probabilities stored as a numpy array
    to a labeled xarray DataArray.

    Parameters
    ----------
    state_probs : numpy.ndarray
        An n_sims by n_cycles + 1 by n_states array storing state
        occupancy probabilities by model cycle.

    Returns
    -------
    xarray.DataArray
        State occupancy probabilities stored in an array with dimensions
        "sim", "time", and "state", indexing parameter simulations from the
        PSA, model time (in years), and the health state, respectively.
    """
    return xr.DataArray(
        state_probs,
        dims=["sim", "time", "state"],
        coords={  # We assumed each cycle is a year
            "sim": range(state_probs.shape[0]),
            "time": range(state_probs.shape[1]),
            "state": ["Sick", "Sicker", "Death"],
        },
    )
```

```
>>> state_probs_psa = hta.label_state_probs(state_probs_psa)
>>> print(state_probs_psa)
<xarray.DataArray (sim: 3, time: 6, state: 3)>
array([[[1.        , 0.        , 0.        ],
        [0.74729753, 0.        , 0.        ],
        [0.5584536 , 0.        , 0.        ],
        [0.41733099, 0.        , 0.        ],
        [0.31187042, 0.        , 0.        ],
        [0.23305999, 0.        , 0.        ]],

       [[1.        , 0.        , 0.        ],
        [0.81609223, 0.        , 0.        ],
        [0.66600653, 0.        , 0.        ],
        [0.54352276, 0.        , 0.        ],
        [0.4435647 , 0.        , 0.        ],
        [0.36198971, 0.        , 0.        ]],

       [[1.        , 0.        , 0.        ],
        [0.81540843, 0.        , 0.        ],
        [0.66489091, 0.        , 0.        ],
        [0.54215765, 0.        , 0.        ],
        [0.44207992, 0.        , 0.        ],
        [0.36047569, 0.        , 0.        ]]])
Coordinates:
  * sim      (sim) int64 0 1 2
  * time     (time) int64 0 1 2 3 4 5
  * state    (state) <U6 'Sick' 'Sicker' 'Death'
>>> print(state_probs_psa.mean(dim="sim"))
<xarray.DataArray (time: 6, state: 3)>
array([[1.        , 0.        , 0.        ],
       [0.79293273, 0.        , 0.        ],
       [0.62978368, 0.        , 0.        ],
       [0.5010038 , 0.        , 0.        ],
       [0.39917168, 0.        , 0.        ],
       [0.31850847, 0.        , 0.        ]])
Coordinates:
  * time     (time) int64 0 1 2 3 4 5
  * state    (state) <U6 'Sick' 'Sicker' 'Death'
```

Good practices with both R and Python

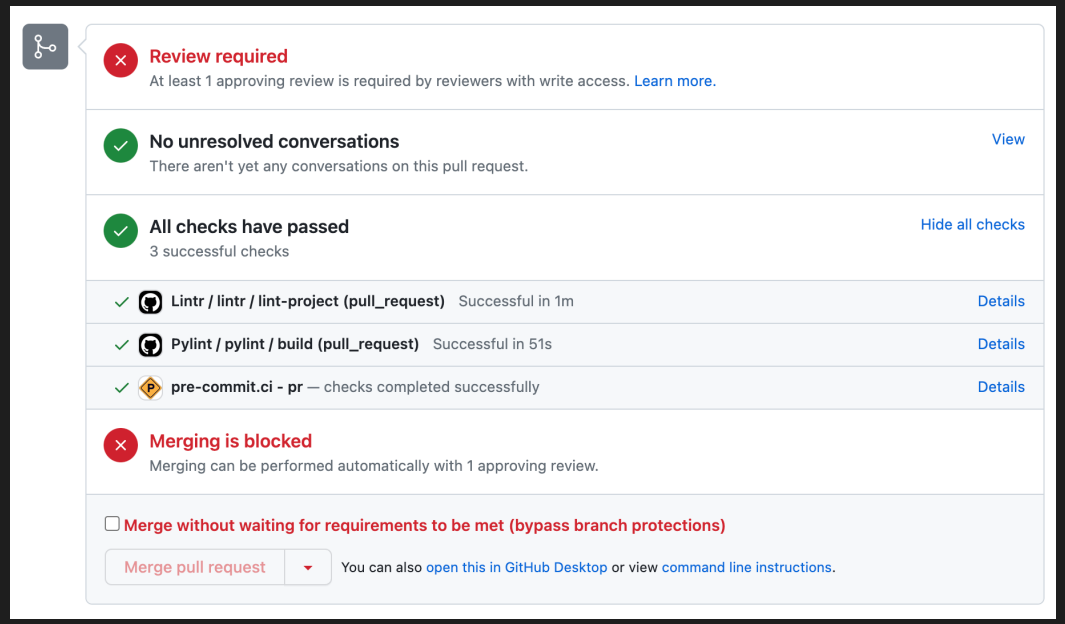
	Python	R
Create websites to view the documentation of your functions and classes	sphinx	pkgdown
Use a process to manage software dependencies	poetry	renv
Use a linter to enforce a coding style	pylint	lintr
Use a styler to automatically modify bad style	black	styler
Use pre-commit to catch bad style, files, commit messages, etc. before making a commit	pre-commit	
Unit test your code to ensure it is correct and that modifications do not have unintended consequences	pytest	testthat
Use continuous integration to automate style checks and unit tests before merging to your “main” branch	GitHub Actions	

Good practices with Python: pylint, GitHub actions, pre-commit, and pytest

```
(hta-py3.11) (base) devin@devins-mbp-2 r-hta-2023 % pylint aa_run.py
************* Module aa_run
aa_run.py:17:0: C0103: Constant name "sample_size" doesn't conform to UP
PER_CASE naming style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 9.38/10 (previous run: 10.00/10, -0.62)
```

```
(hta-py3.11) (base) devin@devins-mbp-2 r-hta-2023 % git commit -m "refactor: minor formatting edits for the slides"
trim trailing whitespace.....................................Passed
fix end of files.............................................Passed
check yaml...........................................(no files to check)Skipped
check for added large files..................................Passed
black........................................................Passed
mdformat.............................................(no files to check)Skipped
style-files..................................................Passed
lintr........................................................Passed
readme-rmd-rendered..................................(no files to check)Skipped
parsable-R...................................................Passed
no-browser-statement.........................................Passed
check for added large files..................................Passed
fix end of files.............................................Passed
Don't commit common R artifacts......................(no files to check)Skipped
pylint.......................................................Passed
poetry-check.........................................(no files to check)Skipped
```




```
(hta-py3.11) (base) devin@devins-mbp-2 r-hta-2023 % pytest tests
========================================================================= test session starts
platform darwin -- Python 3.11.0, pytest-7.3.1, pluggy-1.0.0
rootdir: /Users/devin/code/r-hta-2023
collected 3 items

tests/test_r_hta.py ..F

=============================================================================== FAILURES ===
_____________________________________________________________________________ test_dummy ____

    def test_dummy():
        """Dummy test that can be tweaked to show an example of a failing test."""
>       assert 2 + 2 == 3
E       assert (2 + 2) == 3
```