easy PEASY decision trees in or the Excel user

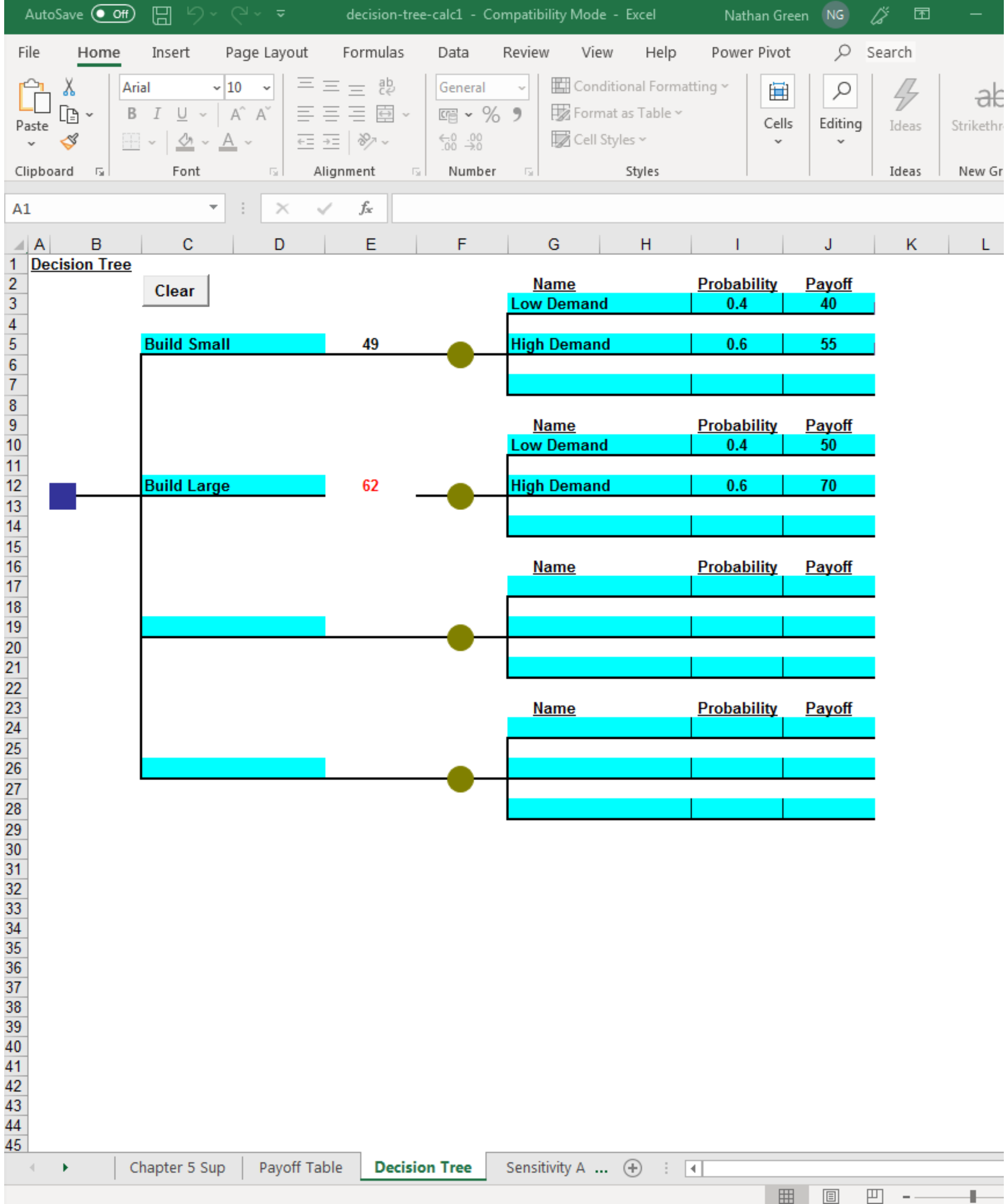
Nathan Green

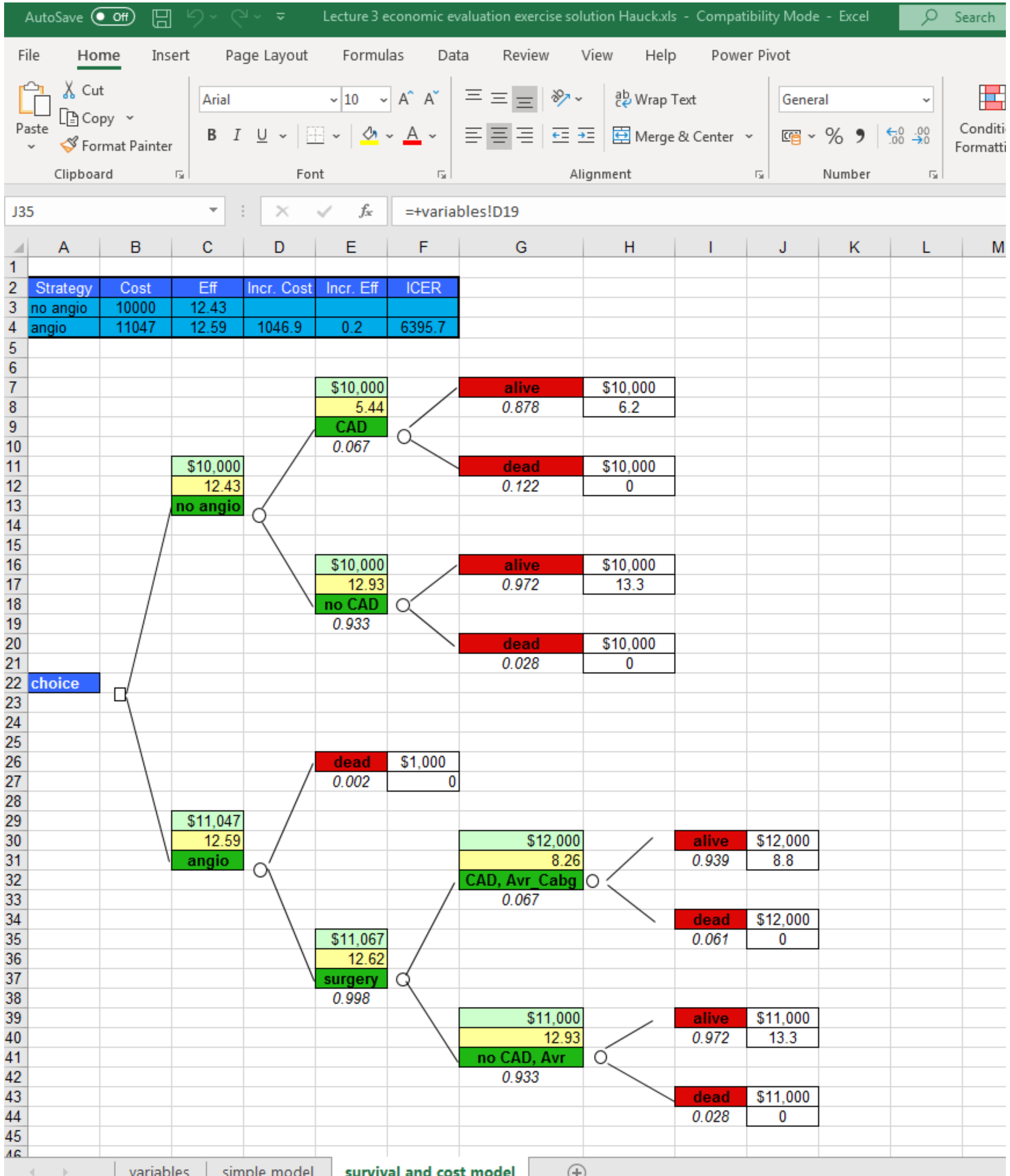
Imperial College London

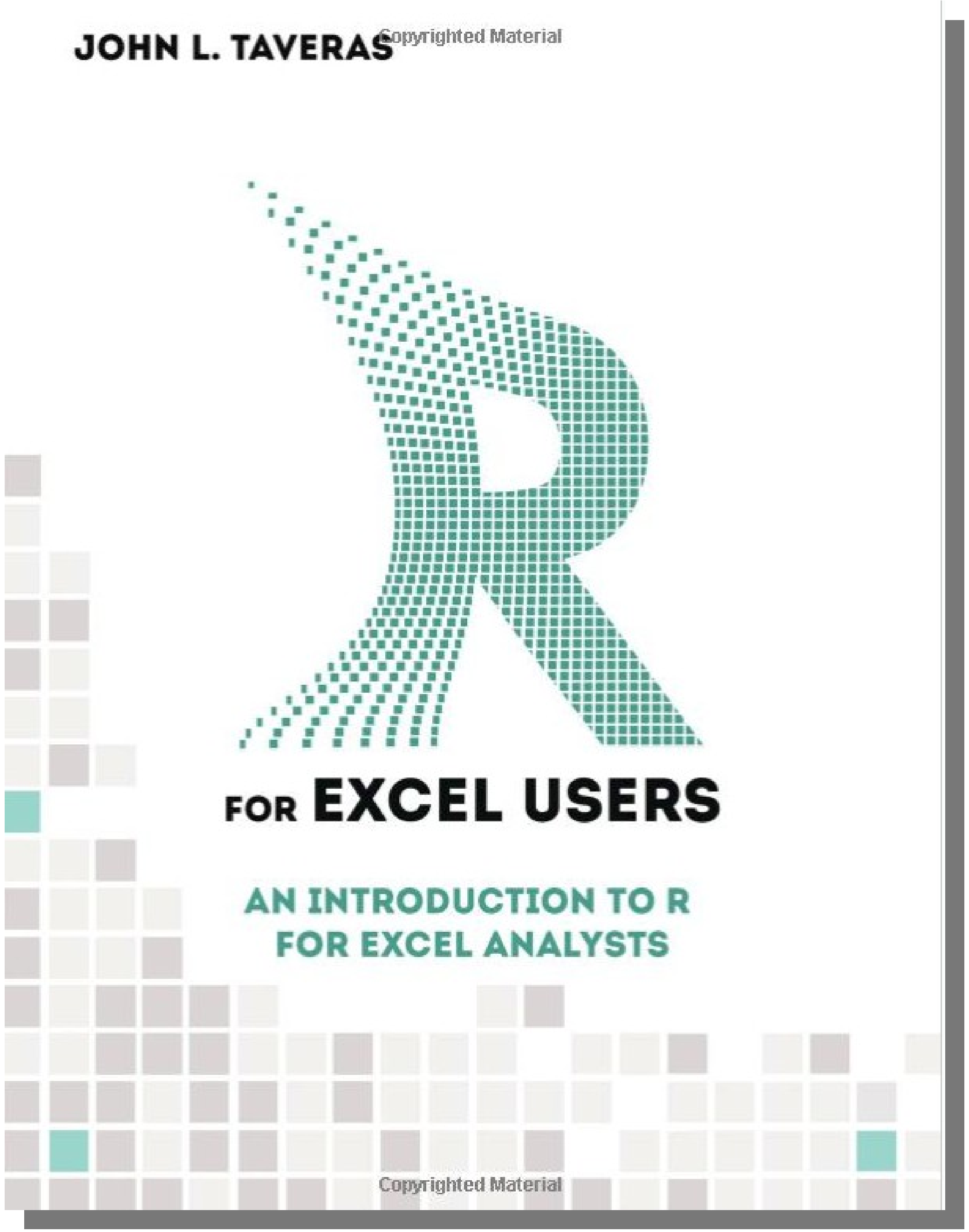
9th July 2019

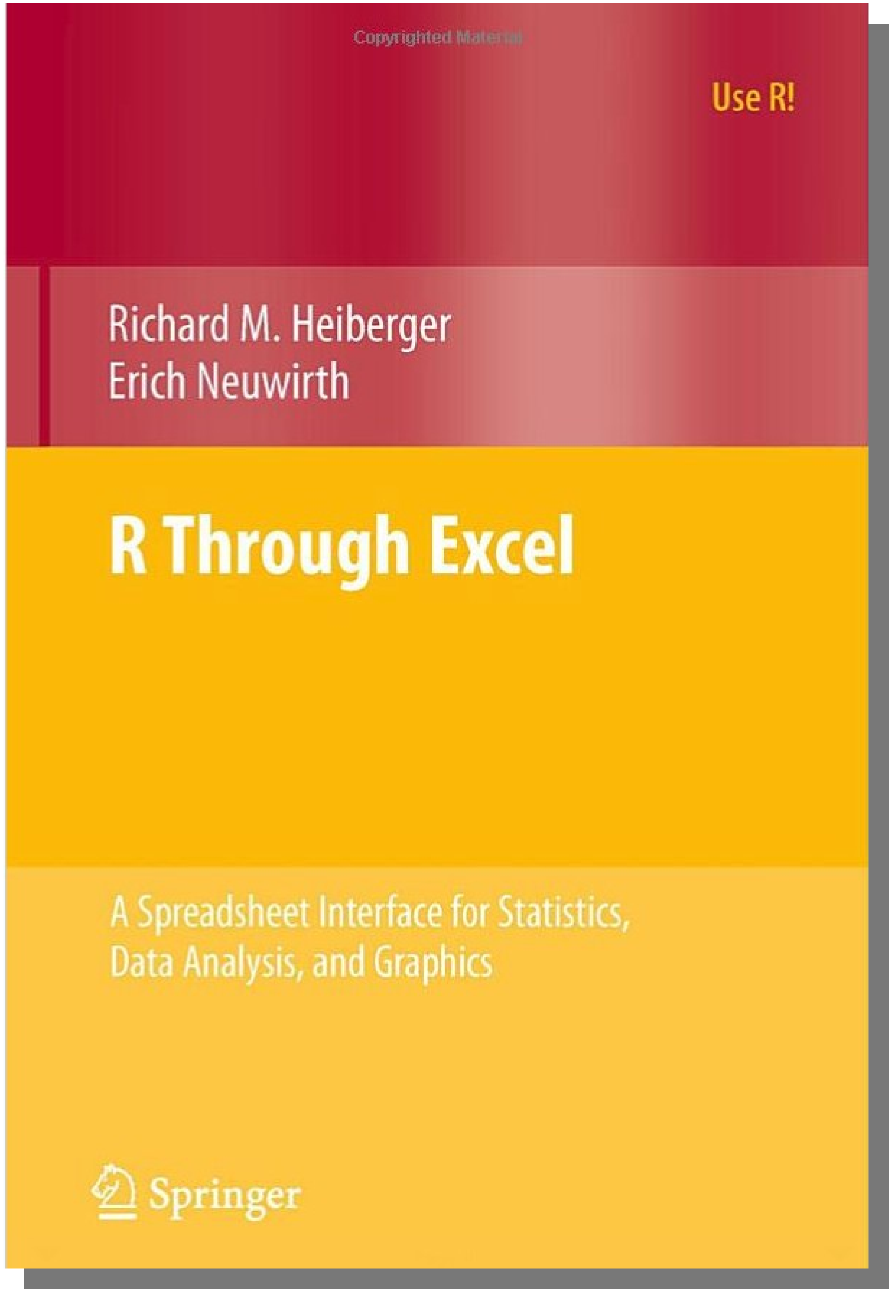
What are decision trees?

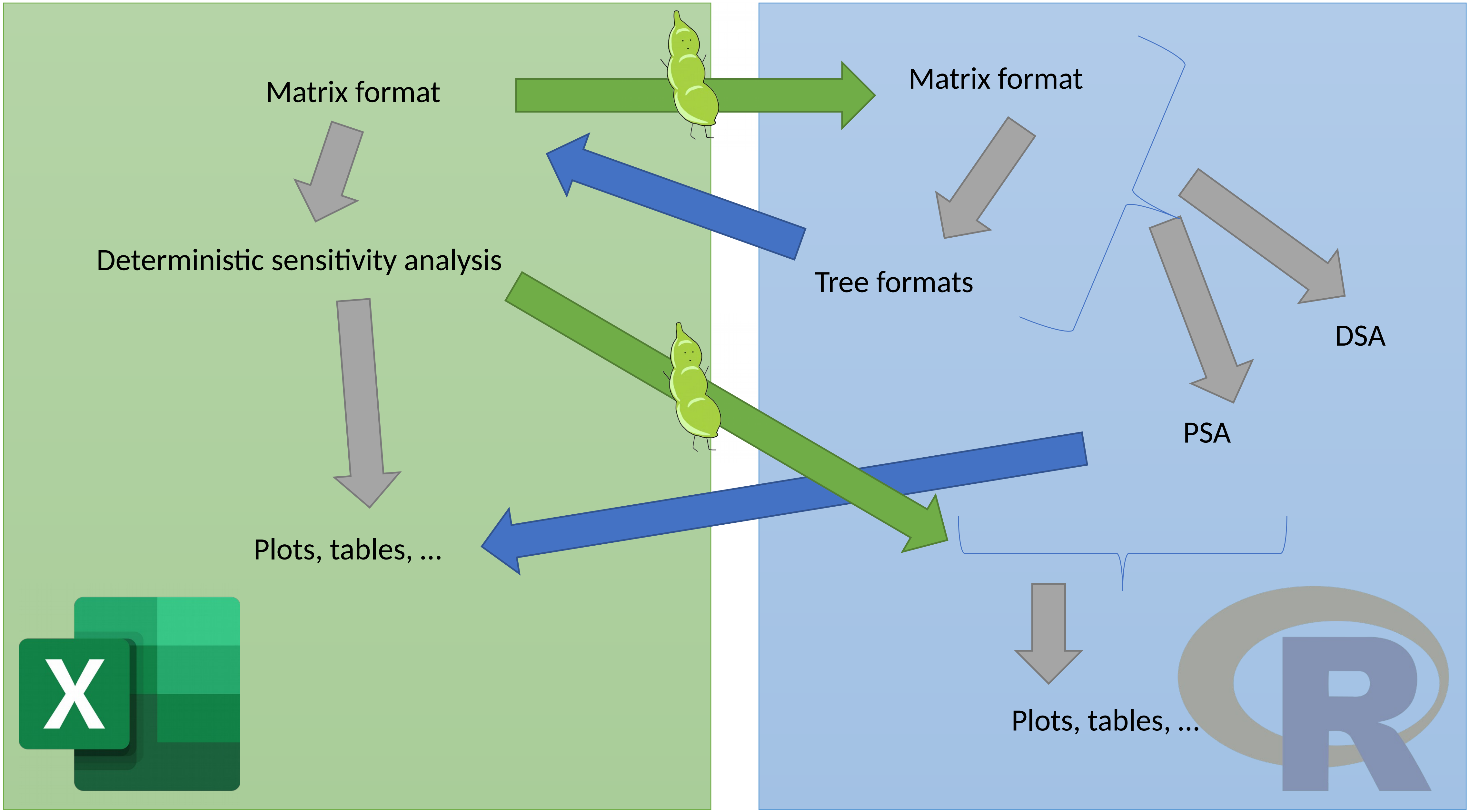
- Can’t see the trees for the forest; No, not *those* decision trees!
- Diagrammatic representations of decision analysis process
- *Decisions* are *square nodes*, a point where several alternatives are possible.
- A *chance node*, typically represented by a circle, is a point in a decision tree where chance determines which event will occur.
- The sum of probabilities for all branches emanating from a chance node must equal 1, because one of the events must occur.


decision-tree-calc1 - Compatibility Mode - Excel
Decision Tree
Clear
Build Small
49
Build Large
62
Name
Probability
Payoff
Low Demand
0.4
40
High Demand
0.6
55
Low Demand
0.4
50
High Demand
0.6
70
Chapter 5 Sup
Payoff Table
Decision Tree
Sensitivity A


Lecture 3 economic evaluation exercise solution Hauck.xls - Compatibility Mode - Excel
=+variables!D19
Strategy
Cost
Eff
Incr. Cost
Incr. Eff
ICER
no angio
10000
12.43
angio
11047
12.59
1046.9
0.2
6395.7
choice
$10,000
12.43
no angio
$10,000
5.44
CAD
0.067
alive
0.878
$10,000
6.2
dead
0.122
$10,000
0
$10,000
12.93
no CAD
0.933
alive
0.972
$10,000
13.3
dead
0.028
$10,000
0
$11,047
12.59
angio
dead
0.002
$1,000
0
$11,067
12.62
surgery
0.998
$12,000
8.26
CAD, Avr Cabg
0.067
alive
0.939
$12,000
8.8
dead
0.061
$12,000
0
$11,000
12.93
no CAD, Avr
0.933
alive
0.972
$11,000
13.3
dead
0.028
$11,000
0
variables
simple model
survival and cost model


JOHN L. TAVERAS
Copyrighted Material
R
FOR EXCEL USERS
AN INTRODUCTION TO R
FOR EXCEL ANALYSTS
Copyrighted Material


Copyrighted Material
Use R!
Richard M. Heiberger
Erich Neuwirth
R Through Excel
A Spreadsheet Interface for Statistics,
Data Analysis, and Graphics
Springer


Matrix format
Deterministic sensitivity analysis
Plots, tables, …
Matrix format
Tree formats
DSA
PSA
Plots, tables, …

CEdecisiontree

build canceled | build unknown | codecov unknown | stability experimental

An R package for lightweight cost-effectiveness analysis using decision trees.

Requests and comments welcome; please use Issues.

Installing CEdecisiontree

To install the development version from github:

```
library(devtools)
install_github("Health-Economics-in-R/CEdecisiontree")
```

Then, to load the package, use:

```
library(CEdecisiontree)
```

Motivation

Decisions trees can be modelled as special cases of more general models using available packages in R e.g. heemod, mstate or msm. Further, full probabilty models could be fit using a Bayesian model with e.g. jags or WinBUGS. However, simple decision tree models are often built in Excel, using statistics from literature or expert knowledge. This package is a analogue to these, such that models can be specified in a very similar and simple way.

Calculation

A decision tree is defined by parent-child pairs, i.e. from-to connections, and the probability and associated value (e.g. cost) of traversing each of the connections. Denote the probability of transitioning from node i to j as p_{ij} and the cost attributable to node i as c_i. Where no connection exists between two nodes we shall say that the parent's set of children is the empty set $\emptyset$. Denote the set of children by $child(\cdot)$. Clearly, there are no p_{ij} or c_j in this case but for computational purposes we will assume that $p_{ij} = NA$ and $c_j = 0$.

The expected value at each node $i \in S$ is calculated by 'folding back' using the recursive formula

$$\hat{c}_i = c_i + \sum_{j \in child(i)} p_{ij} \hat{c}_j$$

Expected cost formula

A decision tree is defined by parent-child pairs, i.e. from-to connections, and the probability and associated value (e.g. cost) of traversing each of the connections. Denote the probability of transitioning from node i to j as p_{ij} and the cost attributable to node i as c_i. Where no connection exists between two nodes we shall say that the parent's set of children is the empty set $\emptyset$. Denote the set of children by $child(\cdot)$. Clearly, there are no p_{ij} or c_j in this case but for computational purposes we will assume that $p_{ij} = NA$ and $c_j = 0$.

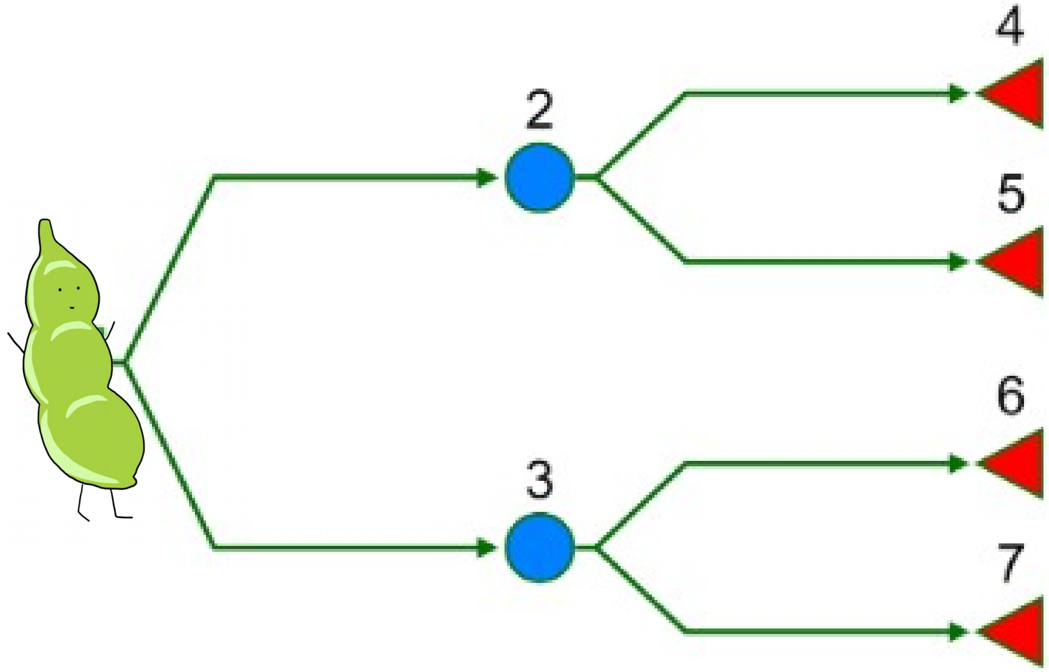
The expected value at each node $i \in S$ is calculated by 'folding back' using the recursive formula

$$\hat{c}_i = c_i + \sum_{j \in child(i)} p_{ij} \hat{c}_j$$

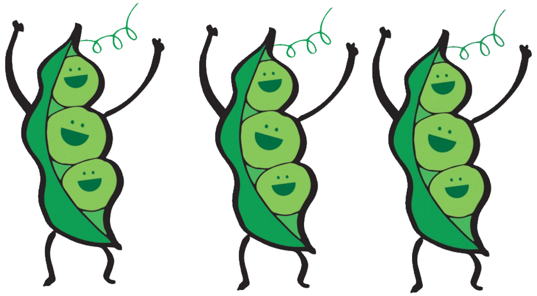
with boundary values at the terminal nodes

$$\hat{c}_i = c_i \text{ for } i = \{S : child(s) = \emptyset\}.$$

Example



Input formats: matrix

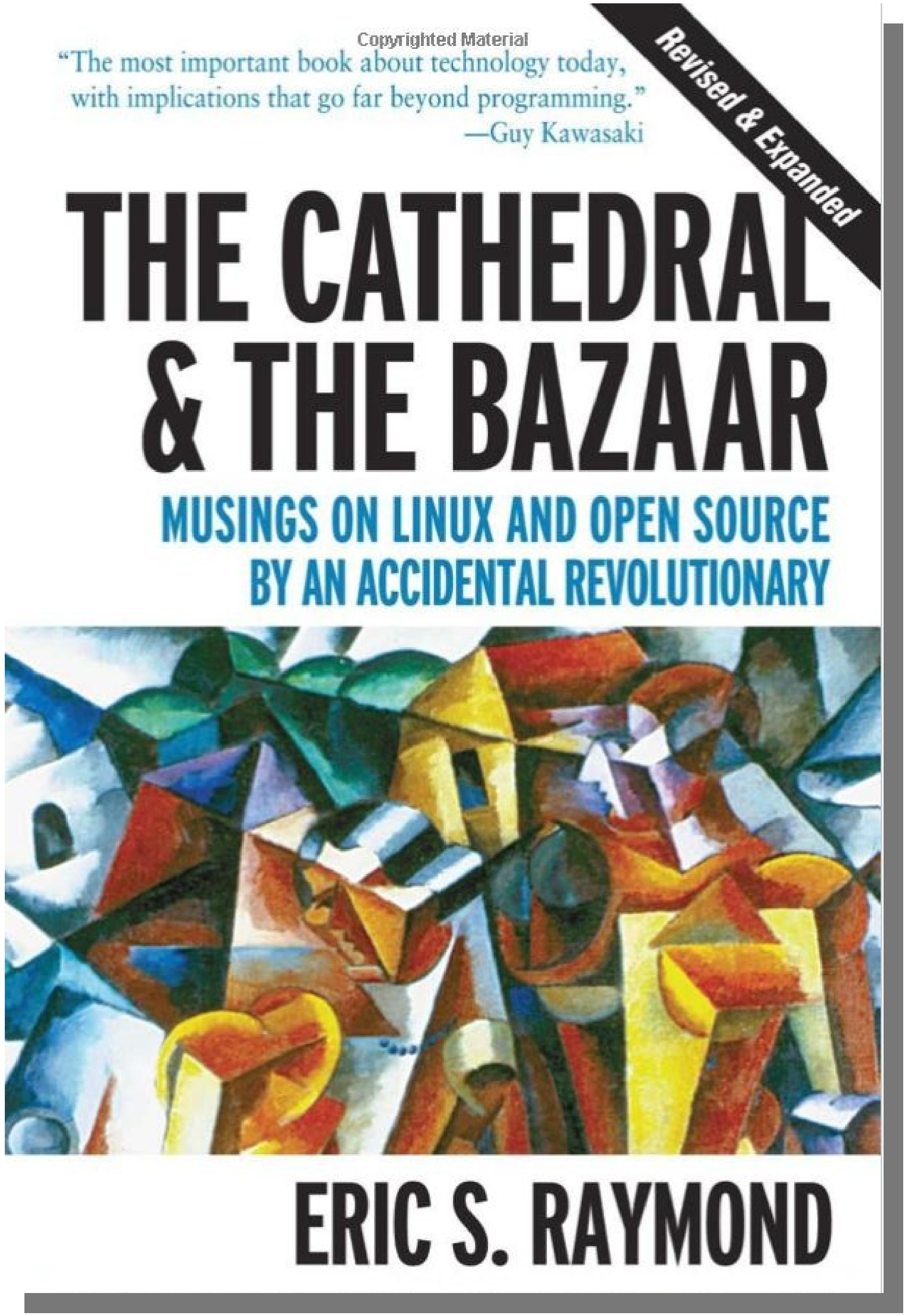


```
cost
#> # A tibble: 3 x 7
#>     `1`   `2`   `3`   `4`   `5`   `6`   `7`
#>   <dbl> <int> <int> <int> <int> <int> <int>
#> 1    NA    10     1    NA    NA    NA    NA
#> 2    NA    NA    NA    10     1    NA    NA
#> 3    NA    NA    NA    NA    NA    10     1
```

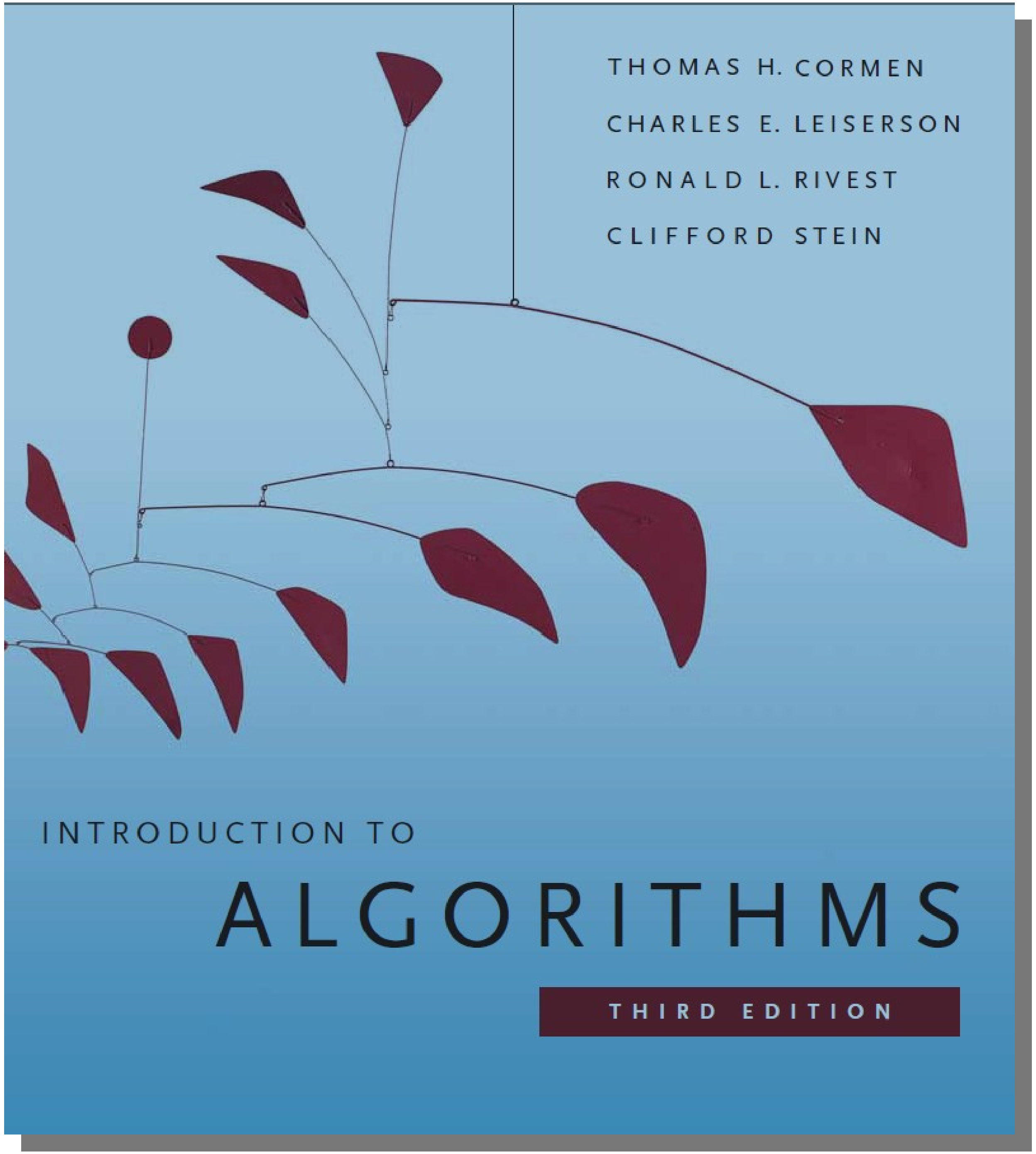
```
probs
#> # A tibble: 3 x 7
#>     `1`   `2`   `3`   `4`   `5`   `6`   `7`
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1    NA   0.2   0.8 NA    NA    NA    NA
#> 2    NA NA    NA      0.2   0.8 NA    NA
#> 3    NA NA    NA    NA    NA      0.2   0.8
```

9. Smart data structures and dumb code works a lot better than the other way around.

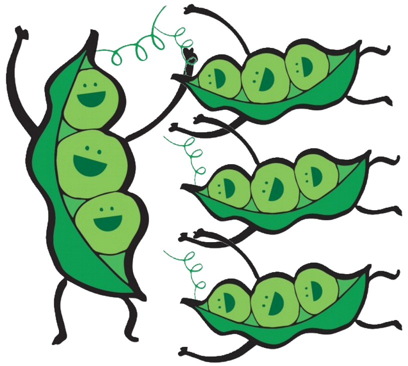
12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.




Algorithms
FOURTH EDITION
ROBERT SEDGEWICK | KEVIN WAYNE


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION

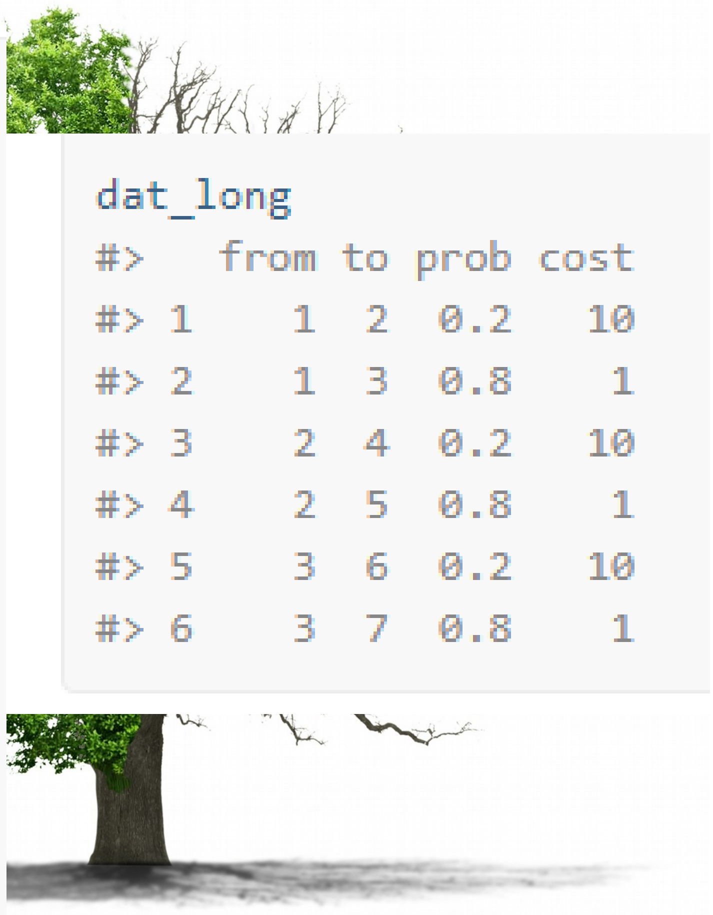
Data structures



- A way to store and organize data in order to **facilitate access and modifications**
- No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them
- Dynamic programming
  - Allows for divide and conquer recurrence
  - Essentially a trade-off of space for time
  - Repeatedly recomputing a given quantity is harmless unless the time spent doing so becomes a drag on performance then better off storing the results of the initial computation and looking them up instead of recomputing them again.
    - E.g. Fibonacci numbers, Binomial coefficients

Data formats: long array

```r
probs_long <-
  probs %>%
  mutate('from' = rownames(.)) %>%
  melt(id.vars = "from",
       variable.name = 'to',
       value.name = 'prob') %>%
  na.omit()

cost_long <-
  cost %>%
  mutate('from' = rownames(.)) %>%
  melt(id.vars = "from",
       variable.name = 'to',
       value.name = 'cost') %>%
  na.omit()

dat_long <-
  merge(probs_long,
        cost_long)
```



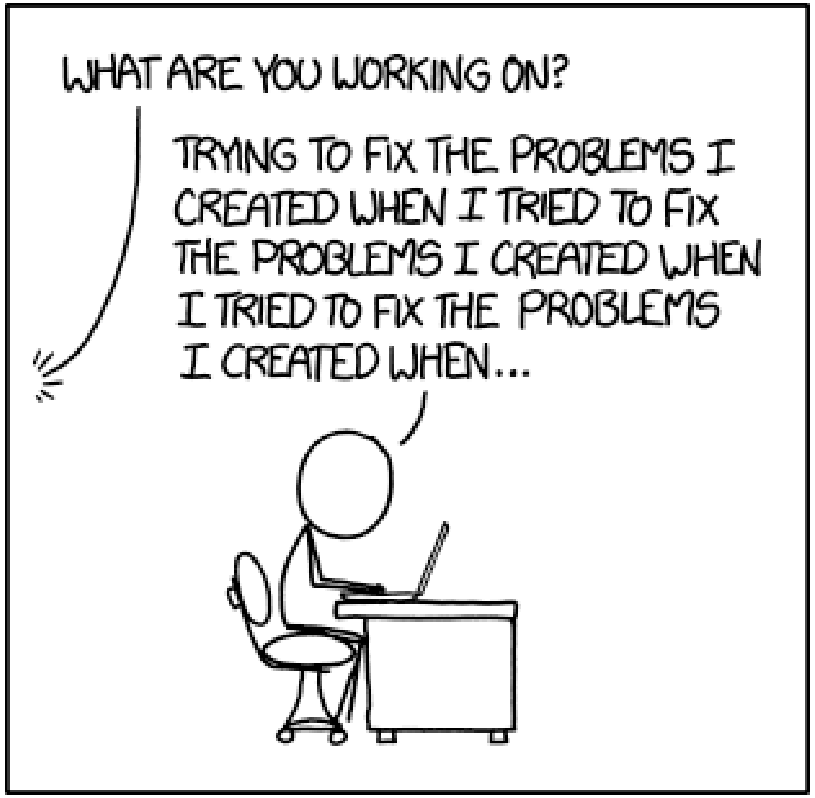
```r
dat_long
#>   from to prob cost
#> 1    1  2  0.2   10
#> 2    1  3  0.8    1
#> 3    2  4  0.2   10
#> 4    2  5  0.8    1
#> 5    3  6  0.2   10
#> 6    3  7  0.8    1
```

Data format: parent-child str

```r
tree <-
 list("1" = c(2,3),
      "2" =  c(4,5),
      "3" =  c(6,7),
      "4" =  c(),
      "5" =  c(),
      "6" =  c(),
      "7" =  c())
dat <-
 data.frame(node = 1:7,
            prob = c(NA, 0.2, 0.8, 0.2, 0.8, 0.2, 0.8),
            vals = c(0,10,1,10,1,10,1))
```

```r
tree
#> $`1`
#> [1] 2 3
#>
#> $`2`
#> [1] 4 5
#>
#> $`3`
#> [1] 6 7
#>
#> $`4`
#> NULL
#>
#> $`5`
#> NULL
#>
#> $`6`
#> NULL
#>
#> $`7`
#> NULL
dat
#>   node prob vals
#> 1    1   NA    0
#> 2    2  0.2   10
#> 3    3  0.8    1
#> 4    4  0.2   10
#> 5    5  0.8    1
#> 6    6  0.2   10
#> 7    7  0.8    1
```

Why tree data structure?



Why tree data structure?

From this

```
for (i in num_from_nodes:1) {

  total <- 0
  for (j in 1:num_to_nodes) {

    if (!is.na(p[i,j])) {

      total <- total + p[i,j]*c_hat[j]
    }
  }
  c_hat[i] <- total + c_hat[i]

}
```

Why tree data structu

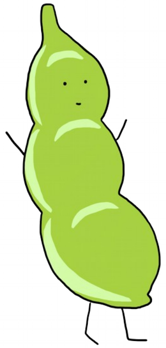
From this

```
for (i in num_from_nodes:1) {


  total <- 0
  for (j in 1:num_to_nodes) {

    if (!is.na(p[i,j])) {

      total <- total + p[i,j]*c_hat[j]
    }
  }
  c_hat[i] <- total + c_hat[i]

}
```

To this:

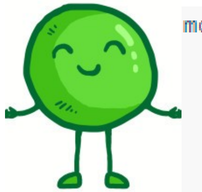
```
dectree_expected_recursive <- function(node,
                                       tree,
                                       dat) {

  if (is.na(node)) {
    return(0)
  }

  c_node <- dat$vals[dat$node == node]

  child <- tree[[node]]

  if (is.null(child)) {
    return(c_node)
  } else {

    pL <- dat$prob[dat$node == child[1]]
    pR <- dat$prob[dat$node == child[2]]

    if (any(is.na(pL))) pL <- 0
    if (any(is.na(pR))) pR <- 0

    return(c_node +
             pL*dectree_expected_recursive(child[1], tree, dat) +
             pR*dectree_expected_recursive(child[2], tree, dat))
  }
}
```

Function dispatch: S3



```
model_transmat <-
  define_model(transmat =
                 list(prob = matrix(data = c(NA, 0.5, 0.5), nrow = 1),
                      vals = matrix(data = c(NA, 1, 2), nrow = 1)
                 ))
model_transmat
#> $prob
#>      [,1] [,2] [,3]
#> [1,]   NA  0.5  0.5
#>
#> $vals
#>      [,1] [,2] [,3]
#> [1,]   NA    1    2
#>
#> attr(,"class")
#> [1] "transmat" "list"
```

```
model_tree <-
  define_model(tree_dat =
                 list(child = list("1" = c(2, 3),
                                   "2" = NULL,
                                   "3" = NULL),
                      dat = data.frame(node = 1:3,
                                       prob = c(NA, 0.5, 0.5),
                                       vals = c(0, 1, 2))
                 ))
model_tree
#> $child
#> $child$`1`
#> [1] 2 3
#>
#> $child$`2`
#> NULL
#>
#> $child$`3`
#> NULL
#>
#>
#> $dat
#>   node prob vals
#> 1    1   NA    0
#> 2    2  0.5    1
#> 3    3  0.5    2
#>
#> attr(,"class")
#> [1] "tree_dat" "list"
```



```
model_long <-
  define_model(dat_long = data.frame(from = c(NA,1, 1),
                                     to = c(1, 2, 3),
                                     prob = c(NA, 0.5, 0.5),
                                     vals = c(0, 1, 2)))
model_long
#>   from to prob vals
#> 1   NA  1   NA    0
#> 2    1  2  0.5    1
#> 3    1  3  0.5    2
#> 4    2  3   NA   NA
#> 5    3  3   NA   NA
```

Deterministic sensitivity analysis

```r
p <- c(NA_real_, 0.4, 0.6)

c2 <- c(10, 50, 100)
c3 <- c(5, 40, 150)

c_grid <-
  expand.grid(c2 = c2,
              c3 = c3) %>%
  cbind(c1 = 0L, .) %>%
  t() %>%
  as.data.frame()
```

```r
c_grid
#>     V1 V2  V3 V4 V5  V6  V7  V8  V9
#> c1   0  0   0  0  0   0   0   0   0
#> c2  10 50 100 10 50 100  10  50 100
#> c3   5  5   5 40 40  40 150 150 150
```

```r
child <- list("1" = c(2, 3),
              "2" = NULL,
              "3" = NULL)
```

```r
tree_dat_sa <- list()

for (i in seq_along(c_grid)) {

  tree_dat_sa[[i]] <-
    define_model(
      tree_dat =
        list(child = child,
             dat = data.frame(
               node = names(child),
               prob = p,
               vals = c_grid[[i]])
        ))
}
```

```r
str(tree_dat_sa, 1)
#> List of 9
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
#>  $ :List of 2
#>   ..- attr(*, "class")= chr [1:2] "tree_dat" "list"
```

Probabilistic sensitivity analysis (PSA)

Probabilistic sensitivity analysis

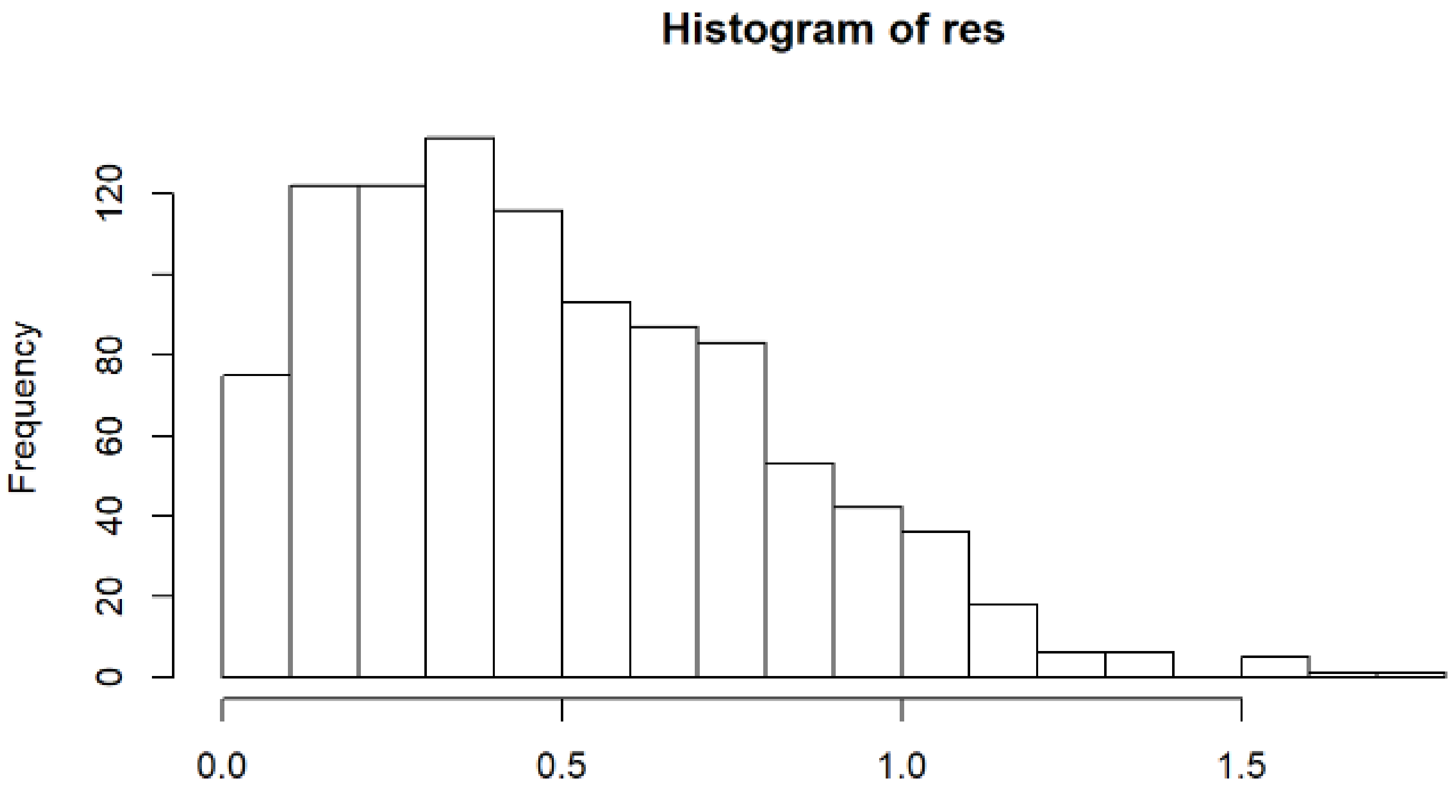
- List columns (see Jenny Brian and purrr package)

```
tree_dat <-
  list(child = list("1" = c(2, 3),
                    "2" = NULL,
                    "3" = NULL),
       dat = tibble(node = 1:3,
                    prob =
                      list(
                        NA_real_,
                        list(distn = "unif", p
                        list(distn = "unif", p
                    vals =
                      list(
                        0L,
                        list(distn = "unif", p
                        list(distn = "unif", p
```

```
tree_dat_sa <- list()

for (i in 1:1000) {

  tree_dat_sa[[i]] <-
    define_model(
      tree_dat =
        list(child = tree_dat$child,
             dat = data.frame(
               node = tree_dat$dat$node,
               prob = lapply(tree_dat$dat$prob, sample_distributions) %>% unlist(),
               vals = lapply(tree_dat$dat$vals, sample_distributions) %>% unlist())
        ))
}
```

```r
head(tree_dat_sa, 2)
#> [[1]]
#> $child
#> $child$`1`
#> [1] 2 3
#>
#> $child$`2`
#> NULL
#>
#> $child$`3`
#> NULL
#>
#>
#> $dat
#>   node       prob      vals
#> 1    1         NA 0.0000000
#> 2    2 0.08081289 0.3373387
#> 3    3 0.93112888 0.7653589
#>
#> attr(,"class")
#> [1] "tree_dat" "list"
#>
#> [[2]]
#> $child
#> $child$`1`
#> [1] 2 3
#>
#> $child$`2`
#> NULL
#>
#> $child$`3`
#> NULL
#>
#>
#> $dat
#>   node     prob      vals
#> 1    1       NA 0.0000000
#> 2    2 0.200610 0.5994206
#> 3    3 0.219152 0.8655122
#>
#> attr(,"class")
#> [1] "tree_dat" "list"
```

```r
res <- map_dbl(tree_dat_sa, dectree_expected_values)
head(res)
#> [1] 0.7399091 0.3099285 0.7204631 0.4817292 0.7451696 0.7709943


hist(res, breaks = 20)
```



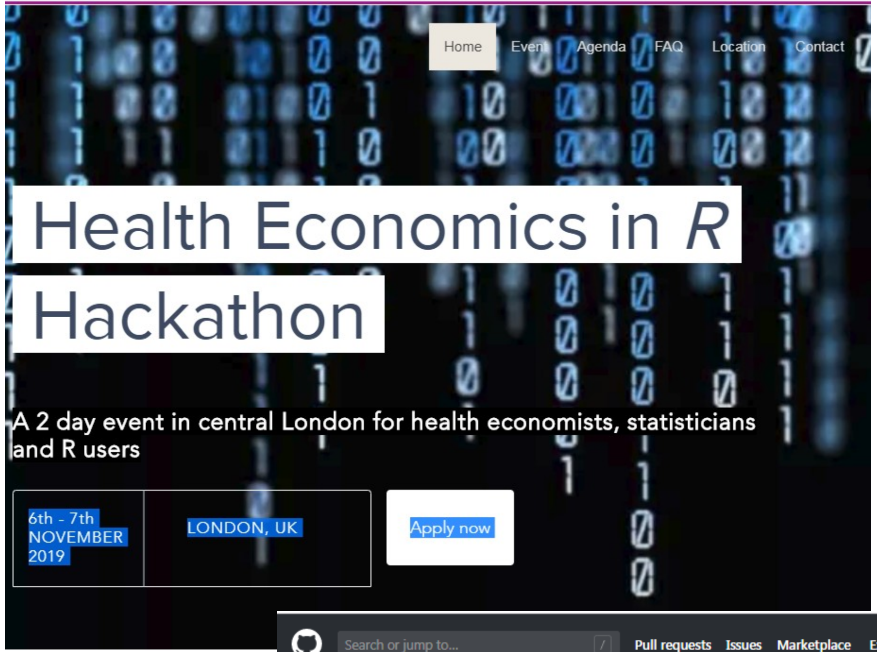
Future

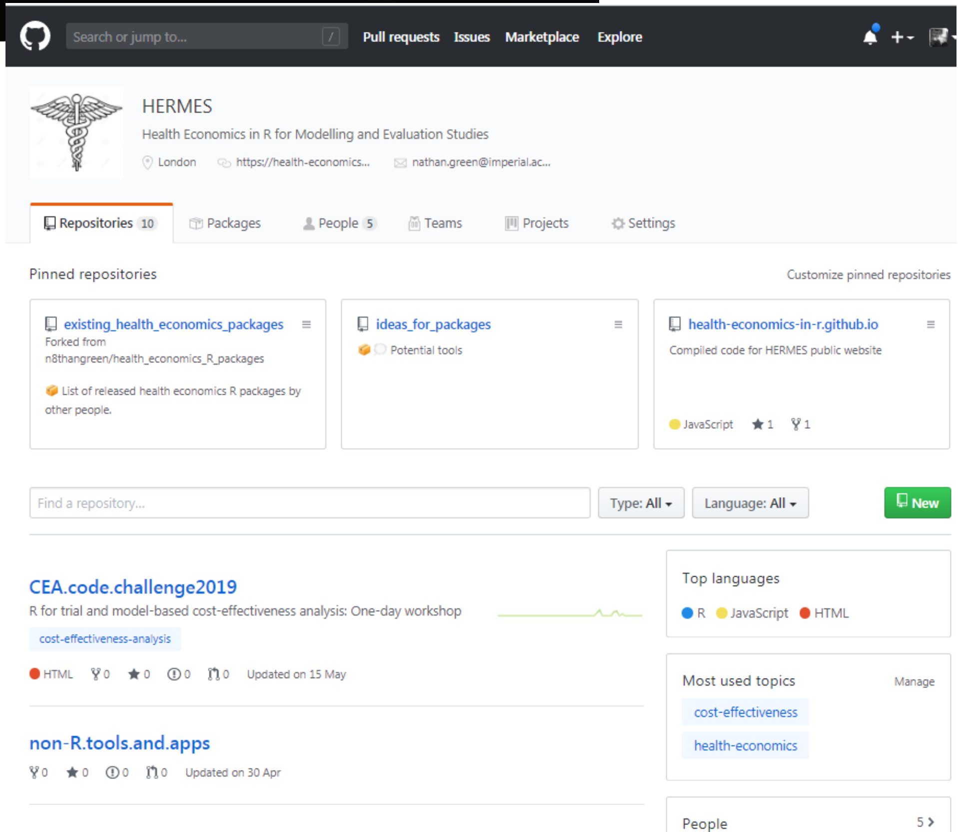
- *data.tree* package

```
#>                                              levelName distn    max    min       p scale shape     type
#> 1  LTBI screening cost                                  unif   0.00   0.00 1.00000    NA    NA  logical
#> 2   ¦--(0,50]                                           unif   0.00   0.00 0.00000                chance
#> 3   ¦   ¦--LTBI                                         unif   0.00   0.00 0.03000                chance
#> 4   ¦   ¦   ¦--Not Agree to Screen                      unif   0.00   0.00 0.40000              terminal
#> 5   ¦   ¦   °--Agree to Screen                          unif 106.00  50.00 0.60000                chance
#> 6   ¦   ¦       ¦--1-Sensitivity                        unif   0.00   0.00 0.16000              terminal
#> 7   ¦   ¦       °--Sensitivity                          unif   0.00   0.00 0.84000                chance
#> 8   ¦   ¦           ¦--Not Start Treatment              unif   0.00   0.00 0.30000              terminal
#> 9   ¦   ¦           °--Start Treatment                  unif   0.00   0.00 0.70000                chance
#> 10  ¦   ¦               ¦--Symptoms Hepatotoxicity     gamma     NA     NA 0.00245 87.889 6.679   chance
#> 11  ¦   ¦               ¦   ¦--Symptoms Nausea         gamma     NA     NA 0.14300     13     5   chance
#> 12  ¦   ¦               ¦   ¦   ¦--Complete Treatment   unif 842.45 511.69 0.80000                chance
#> 13  ¦   ¦               ¦   ¦   ¦   ¦--Effective        unif   0.00   0.00 0.90000              terminal
#> 14  ¦   ¦               ¦   ¦   ¦   °--Not Effective    unif   0.00   0.00 0.10000              terminal
#> 15  ¦   ¦               ¦   ¦   °--Not Complete Treatment unif 140.41  85.24 0.20000              terminal
#> 16  ¦   ¦               ¦   °--Not Symptoms Nausea      unif   0.00   0.00 0.85700                chance
#> 17  ¦   ¦               ¦       ¦--Complete Treatment   unif 842.45 511.69 0.80000                chance
#> 18  ¦   ¦               ¦       ¦   ¦--Effective        unif   0.00   0.00 0.90000              terminal
#> 19  ¦   ¦               ¦       ¦   °--Not Effective    unif   0.00   0.00 0.10000              terminal
#> 20  ¦   ¦               ¦       °--Not Complete Treatment unif 140.41  85.24 0.20000              terminal
#> 21  ¦   ¦               °--Not Symptoms Hepatotoxicity  unif   0.00   0.00 0.99755                chance
#> 22  ¦   ¦                   ¦--Symptoms Nausea         gamma     NA     NA 0.14300     13     5   chance
#> 23  ¦   ¦                   ¦   ¦--Complete Treatment   unif 842.45 511.69 0.80000                chance
#> 24  ¦   ¦                   ¦   ¦   ¦--Effective        unif   0.00   0.00 0.90000              terminal
#> 25  ¦   ¦                   ¦   ¦   °--Not Effective    unif   0.00   0.00 0.10000              terminal
#> 26  ¦   ¦                   ¦   °--Not Complete Treatment unif 140.41  85.24 0.20000              terminal
#> 27  ¦   ¦                   °--Not Symptoms Nausea      unif   0.00   0.00 0.85700                chance
#> 28  ¦   ¦                       ¦--Complete Treatment   unif 842.45 511.69 0.80000                chance
#> 29  ¦   ¦                       ¦   ¦--Effective        unif   0.00   0.00 0.90000              terminal
#> 30  ¦   ¦                       ¦   °--Not Effective    unif   0.00   0.00 0.10000              terminal
#> 31  ¦   ¦                       °--Not Complete Treatment unif 140.41  85.24 0.20000              terminal
```

Future

- Hackathon!
  - 6^{th}-7^{th} November 2019 @ Imperial College London
  - https://n8thangreen.wixsite.com/hermes-hack-london
- GitHub organisation (HERMES)






PEAS &
THANK YOU

- How to become a Hacker
- The Hacker Attitude
- 1. The world is full of fascinating problems waiting to be solved.
- 2. No problem should ever have to be solved twice.
- 3. Boredom and drudgery are evil.
- 4. Freedom is good.
- 5. Attitude is no substitute for competence.

Comparison with heemod

```r
suppressPackageStartupMessages(library(heemod))

mat_base <- define_transition(
  state_names = as.character(1:8),
  0, 0.2, 0.8,0,0,0,0,0,
  0,0,0, 0.2, 0.8,0,0,0,
  0,0,0,0,0, 0.2, 0.8,0,
  0,0,0,0,0,0,0,1,
  0,0,0,0,0,0,0,1,
  0,0,0,0,0,0,0,1,
  0,0,0,0,0,0,0,1,
  0,0,0,0,0,0,0,1
)
```

```r
state_1 <- define_state(
  cost_total = 0,
  qaly = 0)

state_2 <- define_state(
  cost_total = 10,
  qaly = 1)

state_3 <- define_state(
  cost_total = 1,
  qaly = 1)

state_4 <- define_state(
  cost_total = 10,
  qaly = 1)

state_5 <- define_state(
  cost_total = 1,
  qaly = 1)
```

```r
state_6 <- define_state(
  cost_total = 10,
  qaly = 1)

state_7 <- define_state(
  cost_total = 1,
  qaly = 1)

state_8 <- define_state(
  cost_total = 0,
  qaly = 0)

strat_base <- define_strategy(
  transition = mat_base,
  "1" = state_1,
  "2" = state_2,
  "3" = state_3,
  "4" = state_4,
  "5" = state_5,
  "6" = state_6,
  "7" = state_7,
  "8" = state_8
)
```

```r
run_model(
  strat_base,
  cycles = 100,
  cost = cost_total,
  effect = qaly)
#> No named model -> generating names.
#> 1 strategy run for 100 cycles.
#>
#> Initial state counts:
#>
#> 1 = 1000L
#> 2 = 0L
#> 3 = 0L
#> 4 = 0L
#> 5 = 0L
#> 6 = 0L
#> 7 = 0L
#> 8 = 0L
#>
#> Counting method: 'life-table'.
#>
#> Values:
#>
#>   cost_total qaly
#> I       5600 2000
```