```lua
#!/usr/bin/env lua
local help = [[
binr.lua : build rules via stochastic incremental XAI
(c) 2025, Tim Menzies, timm@ieee.org, mit-license.org

Options:
  -h              Show help.
  -e  era=20      Number of rows in an era
  -b  bins=7      Number of bins for discretization.
  -s  seed=42     Random number seed.
  -f  file=../data/auto93.csv  ]]

-- coerce(s) --> v ;; Return int or float or bool or string from `s`.
local function coerce(s)
  if s then return tonumber(s) or s:match'^%s*(.-)%s*$' end end

local the={}; for k,v in help:gmatch("(%S+)=(%S+)") do the[k] = coerce(v) end
math.randomseed(the.seed)

local DATA, NUM, SYM, COLS, clone, adds

--## Lib

local abs,exp,sqrt,log = math.abs, math.exp, math.sqrt, math.log
local floor,max,rand,cos = math.floor,math.max, math.random, math.cos

local say=io.write
local fmt = string.format

-- sort(t,f) --> t ;; Sort `t` using function `f`.
local sort = function(t,f) table.sort(t,f); return t end
-- lt(f) --> f ;; Return a function that sorts `a` and `b` on `f`.
local lt = function(f) return function(a,b) return f(a) < f(b) end end
-- cat(a) --> s ;; Return a string representation of array `a`.
local cat = function(a) return "{".. table.concat(a," ") .."}" end

-- o(v) --> s ;; Return a string representation of `v`.
local function o(v,     list,dict)
  list = function(a,     u)
           for _,v in ipairs(a) do u[1+#u] = o(v) end; return cat(u) end
  dict = function(d,     u)
           for k,v in pairs(d)  do u[1+#u] = fmt(":%s %s", k, o(v)) end
           return cat(sort(u)) end
  return type(v) == "number" and fmt(v%1==0 and "%.0f" or "%.3f", v) or
         type(v) ~= "table" and tostring(v) or (#v>0 and list or dict)(v,{}) end

-- s2a(s) --> a ;; Return array of words from string `s`, split on ",".
local function s2a(s,   a)
  a={}; for s1 in s:gmatch"([^,]+)" do a[1+#a] = coerce(s1) end; return a end

-- csv(file) --> f ;; Iterator that returns rows from `file`.
local function csv(file,    src)
  src = assert(io.open(file))
  return function(    s)
    s = src:read(); if s then return s2a(s) else src:close() end end end

-- shuffle(t) --> t ;; Randomly shuffle the order of elements in `t`.
local shuffle = function(t,    n)
        for m=#t,2,-1 do n=math.random(m); t[m],t[n]=t[n],t[m] end; return t end

-- cut(a0,n,data) --> t,t ;;Split `a0` at `n` (if `data` exists,split that too).
local function cut(a0,n,  data)
  local a1,a2 = {},{}
  for j,v in ipairs(a0) do if j <= n then a1[1+#a1]=v else a2[1+#a2]=v end end
  return data and clone(data,a1),clone(data,a2) or a1,a2 end

-- mode(d) --> v ;; Return the most frequent key in `d`.
local function mode(d,   v,n)
  v,n = nil,0
  for v1,n1 in pairs(d) do if n1>n then v,n=v1,n1 end end
  return v end

-- box_muller(mu,sd) --> n ;; Return a random number from a Gaussian `mu`,`sd`.
local function box_muller(mu,sd)
  return mu + sd * sqrt(-2 * log(rand())) * cos(6.28 * rand()) end

--## Classes

-- DATA(src) --> DATA ;; Create a new DATA, populated with `src`.
function DATA(  src) return adds(src, {n=0,rows={},cols=nil}) end

-- clone(i,src) --> DATA ;; Return a new DATA with same structure as `i`.
function clone(i,  src) return adds(src, DATA{i.cols.names}) end

-- NUM(at,s) --> NUM ;; Create a NUM object to summarize numbers.
function NUM(at,s)
  return {at=at or 0, of=s, n=0, mu=0, m2=0, sd=0, bins={},
          best=(tostring(s) or ""):find"+$" and 1 or 0} end

-- SYM(at,s) --> SYM ;; Create a SYM object to summarize symbols.
function SYM(at,s) return {at=at, of=s, n=0, has={}, bins={}} end

-- COLS(row) --> COLS ;; Create a COLS object from a list of column names.
function COLS(row,    t,x,y,all)
  x,y,all = {},{},{}
  for n,s in ipairs(row) do
    all[n] = (s:match"^[A-Z]" and NUM or SYM)(n,s)
    if not s:match"X$" then
      t = s:find"[+-]$" and y or x
      t[1+#t] = all[n] end end
  return {all=all, x=x, y=y, names=row} end
```

```lua

--## Methods

-- add(i,v) --> v ;; Update `i` with `v` (incrementing by `inc`).
local function add(i,v)
  if v == "?" then return v end
  i.n = i.n + 1
  if i.has then i.has[v] = 1 + (i.has[v] or 0)
  elseif i.mu then
    local d = v - i.mu
    i.mu = i.mu + d / i.n
    i.m2 = i.m2 + d * (v - i.mu)
    i.sd = i.n<2 and 0 or sqrt((max(0,i.m2)/(i.n - 1)))
  elseif i.rows then
    if not i.cols then i.cols = COLS(v) else
      for _,col in pairs(i.cols.all) do add(col, v[col.at]) end
      i.rows[1 + #i.rows] = v end end
  return v end

-- adds(src,it) --> it ;; Update `it` with all items from `src`.
function adds(src, it)
  it = it or NUM()
  if type(src) == "string"
  then for row in csv(src) do add(it,row) end
  else for _,row in pairs(src or {}) do add(it,row) end end
  return it end

-- norm(i,v) --> n ;; Normalize `v` 0..1 using `i`.
local function norm(i,v)
  return (i.has or v=="?") and v or
            1 / (1 + math.exp(-1.7 * (v - i.mu)/(i.sd + 1e-32))) end

-- bin(i,v) --> n ;; Normalize `v` 0..bins-1 using `i`.
local function bin(i,v)
  return (i.has or v=="?") and v or floor( the.bins * norm(i,v)) end

-- disty(i,row) --> n ;; Return distance of `row` to best goal (using Y cols).
local function disty(i,row,     d)
  d=0; for _,y in pairs(i.cols.y) do d= d + (norm(y, row[y.at]) - y.best)^2 end
  return sqrt(d/#i.cols.y)  end

--## Think

local function scoreGet(data,row,    b,n)
  n = 0
  for _,col in pairs(data.cols.x) do
    print(row[col.at])
    b = bin(col, row[col.at])
    print(col.at, o(b))
    n = n + col.bins[b].mu end
  return n end

local function scorePut(data,row,     b,y)
  y = disty(data,row)
  for _,col in pairs(data.cols.x) do
    b = bin(col, row[col.at])
    if b ~= "?" then
      col.bins[b] = col.bins[b] or NUM(col.at, b)
      add(col.bins[b], y) end end end

local function scoreGuess(data,m,rows,    t)
  t = {}
  for n = m,#rows do t[1+#t] = {-scoreGet(data, rows[n]), rows[n]} end
  return sort(t, function(a,b) return a[1] < b[1] end) end

local function score(data,     seen,rows)
  seen = clone(data)
  rows = shuffle(data.rows)
  for m,row in pairs(rows) do
    add(seen, row)
    scorePut(seen, row)
    if m % the.era==0 then
      best = scoreGuess(seen,m+1,rows)[1]
      print(disty(seen, best[2]), best[2]) end end end

--## Demos

local egs={}

egs["-h"] = function(_) print("\n"..help.."\n") end
egs["-s"] = function(n) math.randomseed(n or the.seed); the.seed =n end
egs["--the"] = function(_) print(o(the)) end
egs["--shuffle"] = function(_) print(o(shuffle{10,20,30,40,50})) end

egs["--csv"] = function(_,    n)
  n=1; for row in csv(the.file) do
          if n % 25 == 0 then print(o(row)) end
          n = n + 1 end end

egs["--num"] = function(_,num)
  num=NUM()
  for _=1,1000 do add(num, box_muller(10,5)) end
  print(fmt("%.3f %.3f", num.mu, num.sd)) end

egs["--data"] = function(_)
  for n,col in pairs(DATA(the.file).cols.x) do
  print(n,o(col)) end end

egs["--disty"]= function(_,    data,num)
  data,t = DATA(the.file), {}
  for n,row in pairs(data.rows) do
    if n % 25 == 0 then t[1+#t] = disty(data,row) end end
  print(o(sort(t))) end

egs["--score"] = function(_) score(DATA(the.file)) end

egs["--all"] = function(_,   n)
                 n = the.seed
                 for k,_ in pairs(egs) do
                   math.randomseed(n)
                   if k~="--all" then print("\n-----",k); egs[k]() end end end

-- cli(d,funs) --> nil ;; Update `d` with flags from command-line; run `funs`.
local function cli(d,funs)
  for i,s in pairs(arg) do
    if funs[s]
    then funs[s](coerce(arg[i+1]))
    else for k,_ in pairs(d) do
           if k:sub(1,1)==s:sub(2) then d[k]=coerce(arg[i+1]) end end end end end

```

```lua
if arg[0]:find"binr.lua" then cli(the,egs) end
```