```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=10      Number of rows in an era
  -b  bins=7      Number of bins for discretization.
  -B  Budget=30   Max rows to eval.
  -r  repeats=20 Number of experimental repeats.
  -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,min,max,rand,cos= math.floor,math.min,math.max, math.random,math.cos
local say,fmt = io.write, string.format

-- sort(a,f) --> a;; Sort `a` using function `f`.
local sort = function(a,f) table.sort(a,f); return a end

-- o(v|t) --> 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 u end
  dict=function(d,u)
     for k,v in pairs(d) do u[1+#u]=fmt(":%s %s",k,o(v)) end; return 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
         "{".. table.concat((#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:s) --> 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

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

-- ---------------------------------------------------------------------------
--## Classes

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

-- clone(data, ?src:s|t) --> DATA;; Return a new DATA, copy `data`'s structure.
function clone(data,  src) return adds(src, DATA{data.cols.names}) end

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

-- SYM(  at=0,v="") --> SYM;; Create a SYM object to summarize symbols.
function SYM(at,v) return {at=at, of=v, 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:DATA|NUM|SYM, z:v|t) --> z;; Update `i` with `z`.
local function add(i,z)
  if z == "?" then return z end
  i.n = i.n + 1
  if i.has then i.has[z] = 1 + (i.has[z] or 0)
  elseif i.mu then
    local d = z - i.mu
    i.mu = i.mu + d / i.n
    i.m2 = i.m2 + d * (z - 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(z) else
      for _,col in pairs(i.cols.all) do add(col, z[col.at]) end
      i.rows[1 + #i.rows] = z end end
  return z end

-- adds(src:s|t,it=NUM()) --> 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(num,v) --> n;; Normalize `v` 0..1 using `i`.
local function norm(num,v)
  return 1 / (1 + math.exp(-1.702 * (v - num.mu)/(num.sd + 1e-32))) end

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

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

-- ---------------------------------------------------------------------------
--## Think

-- scoreGet(data,row) -> n;; Score row by sum score of the bins it uses.
local function scoreGet(data,row,    b,n)
  n = 0
  for _,col in pairs(data.cols.x) do
    b = bin(col, row[col.at])
    if b ~= "?" then
      if col.bins[b] then
         n = n + col.bins[b].mu  end end end
  return n end

-- scoreGet(data,row,n) -> nil;; Add a score `n` to each bin used by this row.
local function scorePut(data,row,n,     b,y)
  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], n) end end end

-- scoreGuess(data,m,n,rows) --> t;; sort rows[m] to rows[n] by their guesses
local function scoreGuess(data,rows,m,n,    t, top)
  t = {}
  m = m or 1
  n = n or #rows
  for j = m,n do
    row = rows[j]
    t[1+#t] = {scoreGet(data, row), row} end
  return sort(t, function(a,b) return a[1] < b[1] end) end

-- scoreSeen(data)-->data,n;; collect and print stats for this data
local function scoresSeen(data,      t,m,eps)
  t={}; for m,row in pairs(data.rows) do t[1+#t] = disty(data,row) end
  t=sort(t)
  m=#t//10
  eps = 0.35 * (t[9*m] - t[m])/2.56
  print(fmt("%.2f, %.2f, %.2f, %.2f, %.2f; eps= %.2f",
                        t[m], t[3*m], t[5*m], t[7*m], t[9*m], eps))
  return data,eps end

-- score(data,eps)--> row,n,n;; Guess whata re good rows in data.
local function score(data,eps,     labelled,besty,best,y,n,out)
  besty, labelled = 1e32, clone(data)
  for m,row in pairs(data.rows) do
    if m > the.Budget then break end
    add(labelled, row)
    scorePut(labelled, row, disty(labelled,row))
    if m % the.era==0 then
      best = scoreGuess(labelled,labelled.rows)[1][2]
      y = disty(labelled, best)
      if y < besty - eps then besty,bestRow = y,best end end end
  return best, disty(data, best) end


```

```lua
-- ---------------------------------------------------------------------------
--## 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,t)
  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(_,     t,data,eps,y)
                    data,eps = scoresSeen(DATA(the.file))
                    t={}
                    for n = 1,the.repeats do
                      data.rows  = shuffle(data.rows)
                      _,y,seen = score(data,eps)
                      t[n] = 100*y//1 end
                    print(o(sort(t))) 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

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