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

Options:
  -h              Show help.
  -b  bins=7      Number of bins for discretization.
  -e  era=30      Update model every `era` number of rows.
  -r  ruleMax=3   Max conditions in a rule.
  -s  seed=42     Random number seed.
  -f  file=../lua6/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 max,rand,cos = 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`.
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,
          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={}} end

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

```lua

--## Methods

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

-- sub(i,v) --> v ;; Decrement `v` from `i`.
local function sub(i,v) return add(i,v,-1) 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

-- mid(i) --> v|row ;; Return central tendency of `i`.
local function mid(i)--> a | v;; Exepcted value for `i`.
  if     i.mu   then return i.mu
  elseif i.has  then return mode(i.has)
  elseif i.rows then
    if not i._mid then
      local t={}; for _,col in pairs(i.cols.all) do t[1+#t] = mid(col) end
      i._mid = t end
    return i._mid end 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

-- aha(col,v1,v2) --> n ;; Return distance between `v1` and `v2`.
local function aha(col,v1,v2)
  if v1=="?" and v2=="?" then return 1 end
  if col.has then return v1==v2 and 0 or 1 end
  v1,v2 = norm(col,v1), norm(col,v2)
  v1 = v1 ~= "?" and v1 or (v2 > 0.5 and 0 or 1)
  v2 = v2 ~= "?" and v2 or (v1 > 0.5 and 0 or 1)
  return abs(v1 - v2) end

-- distx(i,row1,row2) --> n ;; Return distance  `row1` to `row2` (using X cols).
local function distx(i,row1,row2,     d)
  d=0; for _,x in pairs(i.cols.x) do d= d + aha(x, row1[x.at],row2[x.at])^2 end
  return sqrt(d/#i.cols.x) 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

-- distys(i,rows) --> rows ;; Sort `rows` by their distance to heaven.
local function distys(i,  rows,      y)
   y = function(row) return disty(i, row) end
   return sort(rows or i.rows, function(r1,r2) return y(r1) < y(r2) end) end

--## Think

-- two(data) --> t ;; Incrementally cluster `data` into `best` and `rest`.
local function two(data)
  local train,test,start,todo,seen,best,rest,d
  shuffle(data.rows)
  train,test = cut(data.rows, data.n//2)
  start,todo = cut(train, 4)
  seen       = clone(data, start)
  best,rest  = cut(distys(seen),2,data)
  d          = function(row,what) return distx(seen, row, mid(what)) end
  for n,row in pairs(todo) do
    if n>256 then break end; --say(".")
    if d(row,best) < d(row,rest) then
      add(seen, add(best, row))  ; --say(best.n)
      if best.n > sqrt(seen.n) then  -- print("-")
        add(rest, sub(best, table.remove( distys(best)))) end end end
  distys(best)
  return {best=best, rest=rest, seen=seen, test=test,
          model = lt(function(row) return d(row,best) - d(row,rest) end)} end
```

```lua

--## Demos

local egs={}

egs["-h"] = function(_) print("\n"..help.."\n") end
egs["-s"] = function(n) math.randomseed(n); the.seed =n end
egs["--the"] = function(_) print(o(the)) end
egs["--csv"] = function(_) for row in csv(the.file) do print(o(row)) end end
egs["--shuffle"] = function(_) print(o(shuffle{10,20,30,40,50})) end
egs["--mode"] = function(_)  print(mode{d=2,f=10,g=1})  end

egs["--cut"] = function(_,    b,c)
  b,c=cut({10,20,30,40,50},2); print(o(b),o(c))
  for _ =1,100 do b,c=cut({10,20,30,40,50},2) 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["--distx"]= function(_,    data,t,u)
  data = DATA(the.file)
  t,rows = {}, shuffle(data.rows)
  for n = 2,#rows do t[1+#t] = distx(data,rows[n-1],rows[n]) end
  print(o(sort(t))) end

egs["--disty"]= function(_,    data,num)
  data,t = DATA(the.file), {}
  distys(data)
  for n,row in pairs(data.rows) do t[n]=disty(data,row) end
  print(o(t)) end

egs["--inc"] = function(_, data1,data2)
  data1 = DATA(the.file)
  print(o(mid(data1)))
  data2 = clone(data1)
  for _,row in pairs(data1.rows) do
    add(data2,row)
    if data2.n==50 then print(o(mid(data2))) end end
  while data2.rows do
    sub(data2, table.remove(data2.rows))
    if data2.n==50 then print(o(mid(data2)));break end end end

egs["--two"] = function(_,    data,out,t)
  t,data = {}, DATA(the.file)
  for _=1,20 do
    out = two(data)
    t[1+#t] = (100*disty(out.seen, sort(out.test, out.model)[1]))//1 end
  print(o(sort(t))) 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"two.lua" then cli(the,egs) end
```