```lua
#!/usr/bin/env lua
local _=require("lib")
local the=_.settings[[

L5 : a lean little learning library, in LUA
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua l5.lua [OPTIONS]

OPTIONS:
 -e  --eg       start-up example                       = nothing
 -b  --bins     max number of bins                     = 8
 -d  --dump     on test failure, exit with stack dump = false
 -f  --file     file with csv data                     = ../data/auto93.csv
 -F  --Far      how far to look for poles (max=1)      = .95
 -h  --help     show help                              = false
 -m  --min      min size. If<1 then t^min else min.    = 10
 -n  --nums     number of nums to keep                 = 512
 -p  --p        distance calculation coefficient       = 2
 -r  --rest     size of "rest" set                     = 3
 -s  --seed     random number seed                     = 10019
 -S  --Sample   how many numbers to keep               = 10000 ]]

local any,cli,copy,csv,lt,many,map= _.any,_.cli,_.copy,_.csv,_.lt,_.many,_.map
local o,obj,oo,per,push,rnd,rogues= _.o,_.obj,_.oo,_.per,_.push,_.rnd,_.rogues
local shallowCopy,shuffle,sort    = _.sort,_.shallowCopy,_.sort
local Data,Num,Row,Some,Sym

-- --------------------------------------------------------------------------
Sym=obj"Sym"
function Sym:new(c,x) return {at=c or 0,txt=x or "",n=0,has={}} end
function Sym:add(x)
  if x~="?" then self.n =1+self.n;self.has[x]=1+(self.has[x] or 0) end end
function Sym:discretize(x) return x end
function Sym:dist(v1,v2)
  return  v1=="?" and v2=="?" and 1 or v1==v2 and 0 or 1 end
function Sym:entropy(     e,fun)
  function fun(p) return p*math.log(p,2) end
  e=0; for _,n in pairs(self.has) do if n>0 then e=e-fun(n/self.n) end end
  return e end

-- --------------------------------------------------------------------------
Some=obj"Some"
function Some:new(c,x)
  return {at=c or 0, txt=x or "",n=0,isSorted=true, _has={}} end
function Some:nums()
  if not self.isSorted then table.sort(self._has) end
  self.isSorted=true
  return self._has end

function Some:add(v,    pos)
  if v~="?" then
    self.n=self.n+1
    if #self._has < the.Sample then pos=1+(#self._has)
    elseif math.random()<the.Sample/self.n then pos=math.rand(#self._has) end
    if pos then self.isSorted=false
                self._has[pos]= v end end end

-- --------------------------------------------------------------------------
Num=obj"Num"
function Num:new(c,x)
  return {at=c or 0,txt=x or "",lo=1E32,hi=-1E32, n=0, has=Some(),
          w=(x or ""):find"-$" and -1 or 1} end
function Num:add(x)
  if x~="?" then self.n = self.n+1
                 self.lo = math.min(x,self.lo)
                 self.hi = math.max(x,self.hi)
                 self.has:add(x)  end end
function Num:norm(n,   lo,hi)
  lo,hi=self.lo,self.hi
  return n=="?" and n or (hi-lo < 1E-0 and 0 or  (n-lo)/(hi-lo + 1E-32)) end
function Num:pers(t,    a)
  a=self.has:nums()
  return map(t,function(p) return per(a,p) end) end
function Num:discretize(x,    tmp)
  tmp = (self.hi - self.lo)/(the.bins - 1)
  return self.lo == self.hi and 1 or math.floor(x/tmp+.5)*tmp end

function Num:dist(v1,v2)
  if   v1=="?" and v2=="?" then return 1 end
  v1,v2 = self:norm(v1), self:norm(v2)
  if v1=="?" then v1 = v2<.5 and 1 or 0 end
  if v2=="?" then v2 = v1<.5 and 1 or 0 end
  return math.abs(v1-v2) end

-- --------------------------------------------------------------------------
Row=obj"Row"
function Row:new(data,t) return {cells=t} end
function Row:around(rows)
  return sort(map(rows, function(r) return {row=r,d=self-r} end),lt"d") end
function Row:far(rows)
  return per(self:around(rows),the.far).row end

function Row:__sub(row,    d,n,d1,n1)
  d,n = 0,0
  for i,col in pairs(self._data.cols.x) do
    d1= col:dist(self.cells[col.at], row.cells[col.at])
    n = n + 1
    d = d + d1^the.p end
  return (d/n)^(1/the.p) end

function Row:__lt(row)
  self.evaled, row.evaled = true,true
  local s1,s2,d,n,x,y=0,0,0,0
  local ys = self._data.cols.y
  for _,col in pairs(ys) do
    x,y= self.cells[col.at], row.cells[col.at]
    x,y= col:norm(x), col:norm(y)
    s1 = s1 - 2.71828^(col.w * (x-y)/#ys)
    s2 = s2 - 2.71828^(col.w * (y-x)/#ys) end
  return s1/#ys < s2/#ys end

function Row:discretize()
  self.cooked=map(self._data.cols.all,
                function(col) col:discretize(self.cells[col.at]) end) end


-- --------------------------------------------------------------------------
Data=obj"Data"
function Data:new(src)
  self.rows, self.cols = {}, {names=nil,all={},x={},y={}}
  self:import(src) end

function Data:import(src)
  if   type(src)=="string"
  then csv(src,        function(row) self:add(row) end)
  else map(src or {}, function(row) self:add(row) end) end
  return self end

function Data:clone(  src) return Data({self.cols.names}):import(src) end

function Data:add(row,    id, what)
  function what(c,x)
    return (x:find"^[A-Z]" and Num or Sym)(c,x) end
  if   #self.cols.all==0
  then self.cols.names=row
       for c,x in pairs(row) do
         local col = push(self.cols.all, what(c,x))
         if not x:find":$" then
           push(x:find"[!+-]" and self.cols.y or self.cols.x, col) end end
  else row = row.cells and row or Row(self,row)
       row._data = self
       push(self.rows, row)
       for _,cols in pairs{self.cols.x, self.cols.y} do
         for _,col in pairs(cols) do col:add(row.cells[col.at]) end end end end

function Data:cheat()
  for i,row in pairs(sort(self.rows)) do
    row.rank = math.floor(.5+ 100*i/#self.rows)
    row.evaled = false end
  self.rows = shuffle(self.rows) end

function Data:half(rows,  above,     some,x,y,c,rxs,xs,ys)
  rows = rows or self.rows
  some = many(rows, the.Sample)
  x    = above or any(some):far(some)
  y    = x:far(some)
  c    = x - y
  rxs  = function(r) return {r=r,x=((r-x)^2 + c^2 - (r-y)^2)/(2*c)} end
  xs,ys= {},{}
  for j,rx in pairs(sort(map(rows,rxs),lt"x")) do
    push(j<=#rows/2 and xs or ys, rx.r) end
  return {xs=xs, ys=ys, x=x, y=y, c=c} end

function Data:best(rows,  above,stop)
  rows = rows or self.rows
  stop = stop or (the.min >=1 and the.min or (#rows)^the.min)
  if   #rows < stop
  then return rows
  else local node = self:half(rows,above)
       if    node.x < node.y
       then  return self:best(node.xs, node.x, stop)
       else  return self:best(node.ys, node.y, stop) end end end

function Data:fours(rows, stop)
  rows = rows or shallowCopy(self.rows)
  stop = stop or (the.min >=1 and the.min or (#rows)^the.min)
  if   #rows < stop
  then return rows
  else rows  = shuffle(rows)
       fours = {}; for i=1,4 do push(fours, table.remove(rows)) end
       t = {}
       for row1 in pairs(rows) do
         four1 = sort(map(fours,function(row2)
                                 return {d=row1-row2, r=row2} end),lt"d")[1].r
         t[four1._id] = t[four1._id] or {}
         push(t[four1._id], four1) end end
  self:fours(sort(fours)[1],stop) end

function Data:discretize()
  for _,row in pairs(self.rows) do row:discretize() end end

function Data:xentropy(     e,sym)
  self:discretize()
  e=0
  for _,col in pairs(self.cols.x) do
    sym = Sym()
    for _,row in pairs(self.rows) do sym:add(row.cooked[col.at]) end
    e = e + sym:entropy() end
  return e end


-- --------------------------------------------------------------------------
local eg = {}
local function egs(    fails,old)
  the = cli(the)
  fails=0
  old = copy(the)
  for k,fun in pairs(eg) do
    if the.eg == "all" or the.eg == k then
      for k,v in pairs(old) do the[k]=v end
      math.randomseed(the.seed)
      print("\n>>>>>",k)
      if not fun() then fails = fails+1 end end end
  rogues()
  os.exit(fails) end

function eg.the() oo(the); return true end

function eg.num(  z)
  z=Num(); for i=1,100 do z:add(i) end; print(z); return true end

function eg.sym(  z)
  z=Sym(); for _,x in pairs{1,1,1,1,2,2,3} do z:add(x) end;
  print(z); return true end

function eg.data( d)
  d=Data(the.file);  map(d.cols.x,print) return true end

function eg.dist(    num,d,r1,r2,r3)
  d=Data(the.file)
  num=Num()
  for i=1,20 do
    r1=any(d.rows)
    r2=any(d.rows)
    r3=r1:far(d.rows)
    io.write(rnd(r3-r1)," ")
    num:add(rnd(r2-r1)) end
  oo(sort(num.has:nums()))
  print(#d.rows)
  return true end

function eg.sort(     d)
  d = Data(the.file)
  d:cheat()
  for i=1,#d.rows,32 do print(i,d.rows[i].rank,o(d.rows[i].cells)) end end

function eg.half(     num,tmp)
  num=Num()
  for i=1,20 do
    local d = Data(the.file)
    d:cheat()
    tmp=d:best()
    map(tmp,function(row) num:add(row.rank) end) end
  print(#tmp,o(num:pers{.1,.3,.5,.7,.9}))
  return end

function eg.discretize(   d)
  d=Data(the.file)
  print(d:xentropy()); return true end

function eg.fours(    d)
  d=Data(the.file)
  d:fours() end

egs()
```