```lua
local _=require("lib")
local the=_.settings[[
TINY2: a lean little learning library, in LUA
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua l5.lua [OPTIONS]

OPTIONS:
 -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
 -g  --go      start-up example                       = nothing
 -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,pop,push        =  _.o,_.obj,_.oo,_.per,_.pop,_.push
local rnd,rogues                   = _.rnd,_.rogues
local shallowCopy,shuffle,sort     = _.shallowCopy,_.shuffle,_.sort
local Data,Num,Row,Some,Sym = obj"Data",obj"Num",obj"Row",obj"Some",obj"Sym"

-- Type hints conventions (for function arguments):
--
-- | What            | Notes                                          |
-- |:---------------:|------------------------------------------------|
-- | 2 blanks        | 2 blanks denote optional arguments             |
-- | 4 blanks        | 4 blanks denote local arguments                |
-- | n               | prefix for numerics                            |
-- | s               | prefix for strings                             |
-- | is              | prefix for booleans                            |
-- | fun             | prefix for functions                           |
-- | suffix s        | list of thing (so names is list of strings)    |
-- | xy,row,col,data | for Xys, Rows, Num or Syms, Data               |
--
-- Another convention is that my code starts with a  help string (at top
-- of file) that is parsed to find the settings. Also my code ends with
-- lots of `go.x()` functions that describe various demos. To run
-- these, use `lua tiny2.lua -go x`.

function Row:new(t) --- Hold one record
  return {evaled=false,
          cells=t,
          cooled=shallowCopy(t)} end

function Sym:new(n,s) --- Summarize stream of symbols.
    return {at=n or 0,
            txt=s or "",
            n=0,
            has={}} end

function Some:new(n,s)  --- Keep at most the.Sample numbers
  return {at=n or 0, txt=s or "",n=0, _has={},
          isSorted=true } end

function Num:new(c,x) --- Summarize stream of numbers
  return {at=c or 0,txt=x or "",n=0,
         lo=1E32,hi=-1E32, n=0,
         has=Some(),
          w=(x or ""):find"-$" and -1 or 1} end

function Data:new(src) --- Store rows of data. Summarize the rows in `self.cols`
.
  self.rows, self.cols = {}, {all={},x={},y={}}
  if   type(src)=="string"
  then csv(src,       function(row) self:add(row) end)
  else map(src or {}, function(row) self:add(row) end) end  end
```

```lua
-- ## Row     ----- ----- ---------------------------------------------------------
-- ### sort
function Row:better(row1,row2,data) --- order two rows
  row1.evaled, row2.evaled = true,true
  local s1,s2,d,n,x,y,ys=0,0,0,0
  ys = data.cols.y
  for _,col in pairs(ys) do
    x,y= row1.cells[col.at], row2.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:betters(rows,data) --- order a whole list of rows
  return sort(rows or self.rows,
              function(r1,r2) return self:better(r1,r2,data) end) end

-- #### dist
function Row:dist(row1,row2,data,   tmp,n,d1) -- distance between rows
  tmp,n = 0,0; for i,col in pairs(data.cols.x) do
                 d1     = col:dist(row1[col.at], row2[col.at],data)
                 n, tmp = n + 1,  tmp + d1^the.p end
  return (tmp/n)^(1/the.p) end

function Row:dists(r1,rows,data) --- sort `rows` by distance to `r11.
  return sort(map(rows,
           function(r2) return {r=r2,d=self:dist(r1,r2,data)} end),lt"d") end

function Row:far(row,rows,data) -- Find an item in `rows`, far from `row1.
  return per(self:dists(row,rows,data),the.far).r end

-- ## Sym     ----- ----- --------------------------------------------------------
-- ### update
function Sym:add(s) --- Update.
  if s~="?" then self.n =1+self.n;self.has[s]=1+(self.has[s] or 0) end end

-- ### dist
function Sym:dist(s1,s2) -- Gap between two symbols.
  return  s1=="?" and s2=="?" and 1 or s1==s2 and 0 or 1 end

-- ### query
function Sym:entropy(     e,fun) -- Entropy
  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   ----- ----- --------------------------------------------------------
-- ### update
function Some:add(x,    pos) --- update
  if x~="?" 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]= x end end end

-- ### query
function Some:nums()
  if not self.isSorted then table.sort(self._has) end
  self.isSorted=true
  return self._has end

-- ## Num     ----- ----- ---------------------------------------------------------
-- ### update
function Num:add(n) --- update
  if n~="?" then self.n = self.n+1
                 self.lo = math.min(n,self.lo)
                 self.hi = math.max(n,self.hi)
                 self.has:add(n)  end end

-- ### query
function Num:norm(n,   lo,hi) --- convert `n` to 0..1 for min..max
  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(ns,    a) --- report a list over percentiles
  a=self.has:nums()
  return map(ns,function(p) return per(a,p) end) end

-- ### dist
function Num:dist(n1,n2) --- return 0..1. If unknowns, assume max distance.
  if   n1=="?" and n2=="?" then return 1 end
  n1,n2 = self:norm(n1), self:norm(n2)
  if n1=="?" then n1 = n2<.5 and 1 or 0 end
  if n2=="?" then n2 = n1<.5 and 1 or 0 end
  return math.abs(n1-n2) end
```

```lua
-- ## Data    ----- ----- ---------------------------------------------------------
-- ### create
function Data:body(row) --- Crete new row. Store in `rows`. Update cols.
  row = row.cell and row or Row(row) -- Ensure `row` is a `Row`.
  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

function Data:clone(  src,    data) --- Copy structure. Optionally, add in data.
  data= Data( {map(self.all, function(col) return col.txt end)} )
  map(src or {}, function (row) data:add(row) end)
  return data end

function Data:header(row) --- Create the `Num`s and `Sym`s for the column header
s
  for n,s in pairs(row) do
    local col = push(self.cols.all, (x:find"^[A-Z]" and Num or Sym)(n,s))
    if not s:find":$" then
      push(s:find"[!+-]" and self.cols.y or self.cols.x, col) end end end

-- ### udpate
function Data:add(row) --- the new row is either a header, or a data row
  if #self.cols.all==0 then self:header(row) else self:body(row) end end

-- ### query
function Data:cheat(   ranks) --- return percentile ranks for rows
  for i,row in pairs(self:betters()) do
    row.rank = math.floor(.5+ 100*i/#self.rows) end
  self.rows = shuffle(self.rows)
  return self.rows end

-- ### cluster
function Data:half(  above, --- split data by distance to two distant points
                  some,x,y,c,rxs,xs,ys)
  some= many(self.rows, the.Sample)
     x= above or self:far(any(some),some,data)
     y= self:far(x,some,data)
     c= self:dist(x,y,data)
   rxs=function(r) return
         {r=r,x=(self:dist(r,x,data)^2 + c^2 - self:dist(r,y,data)^2)/(2*c)} end
  xs,ys= self:clone(), self:clone()
  for j,rx in pairs(sort(map(self.rows,rxs),lt"x")) do
    if j<=#self.rows/2 then xs:add(rx.r) else ys:add(rx.r) end end
  return {xs=xs, ys=ys, x=x, y=y, c=c} end

function Data:best(  above,stop,evals) --- recursively divide, looking 4 best le
af
  stop = stop or (the.min >=1 and the.min or (#self.rows)^the.min)
  evals= evals or 2
  if   #self.rows < stop
  then return self,evals
  else local node = self:half(above)
       if    self:better(node.x,node.y)
       then  return node.xs:best(node.x, stop, evals+1)
       else  return node.ys:best(node.y, stop, evals+1) end end end
```

```lua
-- ## Demos/Tests  ----- ----- ----------------------------------------------
local go = {}
function go.the() oo(the); return true end

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

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

function go.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= d:far(r1,d.rows,d)
    io.write(rnd(d:dist(r1,r3,d))," ")
    num:add(rnd(d:dist( r1,r2,d))) end
  oo(sort(num.has:nums()))
  print(#d.rows)
  return true end

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

function go.clone(    d1,d2)
  d1 = Data(the.file)
  d2 = d1:clone(d1.rows)
  oo(d1.cols.x[2])
  oo(d2.cols.x[2]) end

function go.half( d,node)
  d=Data(the.file)
  node = d:half()
  print(#node.xs.rows, #node.ys.rows, d:dist(node.x, node.y,d))end

function go.best(     num)
  num=Num()
  for i=1,20 do
    local d=Data(the.file)
    local _,ranks = d:cheat()
    shuffle(d.rows)
    local leaf,evals = d:best()
    for _,row in pairs(leaf.rows) do num:add(ranks[ row[1] ]) end end
  print(o(num:pers{.1,.3,.5,.7,.9}))
end

function go.bests(     num,tmp)
  num=Num()
  for i=1,20 do
    local d = Data(the.file)
    d:cheat()
    shuffle(d.rows)
    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 go.discretize(   d)
  d=Data(the.file)
  print(d:xentropy()); return true end

function go.four(    num,d,some,evals,ranks)
  num=Num()
  for i=1,20 do
    d=Data(the.file)
    --_,ranks= d:cheat()
    some,evals = d:fours()
    _,ranks = d:cheat()
    print(#some)
    for _,row in pairs(some) do num:add(ranks[row[1]]) end end
  oo(num:pers{.1,.3,.5,.7,.9})
end
-- ## Start  ----- ----- -------------------------------------------------------
local function on(settings,funs,   fails,old)
  fails=0
  old = copy(settings)
  for k,fun in pairs(funs) do
    if settings.go == "all" or settings.go == k then
      for k,v in pairs(old) do settings[k]=v end
      math.randomseed(settings.seed or 10019)
      print("\n>>>>>",k)
      if not fun() then fails = fails+1 end end end
  rogues()
  os.exit(fails) end

the = cli(the)
on(the,go)
```