```lua
--  __                              ___
-- /\ \__     __                   /'___`\
-- \ \ ,_\  /\_\     ___     __  __  /\_\ /\ \
--  \ \ \/  \/\ \  /' _ `\  /\ \/\ \  \/_/// /__
--   \ \ \_  \ \ \ /\ \/\ \ \ \ \_\ \    // /_\ \
--    \ \__\  \ \_\\ \_\ \_\ \/`____ \  /\______/
--     \/__/   \/_/ \/_/\/_/  `/___/> \ \/_____/
--                               /\___/
--                               \/__/

local b4={}; for k,v in pairs(_ENV) do b4[k]=v end;
local help=[[
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, coerce, copy, csv, fmt, gt, lt, many, map, o, obj, oo, per, pop
local push, red, rnd, rogues, settings, shallowCopy, shuffle, sort, the, yellow
function obj(s,    t,i,new)
  t={__tostring = function(x) return s..o(x) end}
  function new(k,...)
    i=setmetatable({},k); return setmetatable(t.new(i,...) or i,k) end
  t.__index = t;return setmetatable(t,{__call=new}) end

local Data,Num,Row,Some,Sym = obj"Data", obj"Num", obj"Row", obj"Some", obj"Sym"

--[[ Type hints conventions:
| Function args    | 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 objects   |

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 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

-- ## Row     ----- ----- -----------------------------------------------------
-- ### sort
function Row:better(row,data) --- order two rows
  self.evaled, row.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= 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

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

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

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

-- ## 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.cols.all, function(col) return col.txt end)} )
  map(data.cols.x,oo)
  map(src or {}, function (row) data:add(row) end)
  return data end

function Data:header(row) --- Create `Num`s and `Sym`s for the column headers
  for n,s in pairs(row) do
    local col = push(self.cols.all, (s: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

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

--- ### dist
function Data:betters(rows,data) --- order a whole list of rows
  return sort(rows or self.rows,
              function(r1,r2) return r1:better(r2,self) end) 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 hunt  for best leaf
  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
-- ## Lib     ----- ----- ------------------------------------------------------------
-- ### Sampling
function any(t) return t[math.random(#t)] end --- select one, at random

function many(t1,n,  t2) --- select `n`
  if n >= #t1 then return shuffle(t1) end
  t2={}; for i=1,n do push(t2, any(t1)) end; return t2 end

function shuffle(t,   j) --- Randomly shuffle, in place, the list `t`.
  for i=#t,2,-1 do j=math.random(i); t[i],t[j]=t[j],t[i] end; return t end

-- ### Maths
function rnd(x, places)
  local mult = 10^(places or 2)
  return math.floor(x * mult + 0.5) / mult end

-- ### Strings
fmt = string.format --- printf clone

local function color(s,n)  --- colorize text
  return fmt("\27[1m\27[%sm%s\27[0m",n,s) end

function red(s)      return color(s,31) end
function yellow(s)   return color(s,34) end

function oo(t) print(o(t)) return t end --- print nested lists
function o(t,    seen,show,u) ---  coerce to string (skip loops, sort slots)
  if type(t) ~=  "table" then return tostring(t) end
  seen=seen or {}
  if seen[t] then return "..." end
  seen[t] = t
  function show(k,v)
    if not tostring(k):find"^_"  then
      v = o(v,seen)
      return #t==0 and fmt(":%s %s",k,v) or o(v,seen) end end
  u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
  if #t==0 then table.sort(u) end
  return "{"..table.concat(u," ")..'}' end

-- ### Lists
function map(t1,fun,    t2)  --- apply `fun` to all of `t1` (skip nil results)
  t2={}; for _,v in pairs(t1) do t2[1+#t2] = fun(v) end; return t2 end

function per(t,p) --- return the pth (0..1) item of `t`.
  p=math.floor(((p or .5)*#t)+.5); return t[math.max(1,math.min(#t,p))] end

function push(t,x) t[1+#t]=x; return x end --- at `x` to `t`, return `x`
function pop(t)    return table.remove(t) end --- remove(and return) last item

function shallowCopy(t)  return copy(t,true) end --- shallow copy of list
function copy(t,  shallow, u) --- copy list
  if type(t) ~= "table" then return t end
  u={}; for k,v in pairs(t) do u[k] = shallow and v or copy(v) end
  return setmetatable(u,getmetatable(t))  end

-- ### Sorting
function sort(t,f) table.sort(t,f); return t end --- sort(and return) list
function lt(x)  return function(t1,t2) return t1[x] < t2[x] end end ---sort <
function gt(x)  return function(t1,t2) return t1[x] > t2[x] end end ---sort >


-- ### Settings
function settings(s,     t) -- create a `the` variable
  t = {_help=s}
  s:gsub("\n [-][%S]+[%s]+[-][-]([%S]+)[^\n]+= ([%S]+)",
         function(k,x) t[k] = coerce(x) end)
  return t end

function coerce(s,    fun) --- Parse `the` config settings from `help`.
  function fun(s1)
    if s1=="true"  then return true end
    if s1=="false" then return false end
    return s1 end
  return math.tointeger(s) or tonumber(s) or fun(s:match"^%s*(.-)%s*$") end

function cli(t) -- Updates from command-line. Bool need no values (just flip)
  for slot,v in pairs(t) do
    v = tostring(v)
    for n,x in ipairs(arg) do
      if x=="-"..(slot:sub(1,1)) or x=="--"..slot then
        v = v=="false" and "true" or v=="true" and "false" or arg[n+1] end end
    t[slot] = coerce(v) end
  if t.help then
    t._help=t._help
            :gsub("%s([-][-]?%S[a-z]*)",function (s) return " "..yellow(s) end)
            :gsub("([A-Z][A-Z]+)"      ,function (s) return red(s) end)
    os.exit(print("\n"..t._help.."\n")) end
  return t end

-- ### csv
function csv(sFilename, fun,      src,s,t) --- call `fun` cells in each CSV line
  src  = io.input(sFilename)
  while true do
    s = io.read()
    if   s
    then t = {}; for s1 in s:gmatch("([^,]+)") do t[1+#t] = coerce(s1) end
         fun(t)
    else return io.close(src) end end end
```

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

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

function go.eg( d)
  d=Data(the.file);  map(d.cols.x,print) 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= r1:far(d.rows,d)
    io.write(rnd(r1:dist(r3,d))," ")
    num:add(rnd(r1:dist( r2,d))) end
  oo(sort(num.has:nums()))
  print(#d.rows) end

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

function go.clone(    d1,d2)
  d1 = Data(the.file)
  d2 = d1:clone(d1.rows); print(1)
  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
```

```lua
-- ## 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 fun()==false then fails = fails+1;print("FAIL!!!!!",k); end end end
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
  os.exit(fails) end

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