```lua
---                                   __
---                                 /'__`\
---   ____     __       ___ ___    /\ \/\ \
---  /',__\  /'__`\   /' __` __`\  \ \ \ \ \
--- /\__, `\/\ \L\.\_ /\ \/\ \/\ \  \ \ \_\ \
--- \/\____/\ \__/.\_\\ \_\ \_\ \_\  \ \____/
---  \/___/  \/__/\/_/ \/_/\/_/\/_/   \/___/
---
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
local l=require"lib0"

local cli,coerce,copy,csv,o,oo = l.cli,l.coerce,l.copy,l.csv,l.o,l.oo
local push,settings            = l.push,l.settings
local Cols, Data, Num, Row, Sym, dist,div,header,mid,norm,row
local the=settings [[
SAM0 : semi-supervised multi-objective explainations
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua eg0.lua [OPTIONS]

OPTIONS:
 -e  --example  start-up example         = ls
 -h  --help     show help                = false
 -p  --p        distance coeffecient     = 2
 -S  --some     how many numbers to keep = 256
 -s  --seed     random number seed       = 10019]]

---- ---- ---- ---- Data
---- ---- ---- Classes
-- Holder of `rows` and their sumamries (in `cols`).
function Data() return {cols=nil,  rows={}} end
-- Hoder of summaries
function Cols() return {klass=nil, names={}, nums={}, x={}, y={}, all={}} end
-- Summary of a stream of symbols.
function Sym(c,s)
  return {n=0,at=c or 0, name=s or "", _has={}} end
-- Summary of a stream of numbers.
function Num(c,s)
  return {n=0,at=c or 0, name=s or "", _has={},
          isNum=true, lo= math.huge, hi= -math.huge, sorted=true,
          w=(s or ""):find"-$" and -1 or 1} end
-- Hold one record
function Row(t) return {cells=t, cooked=copy(t)} end

-- Add one or more items, to `col`.
function adds(col,t) for _,v in pairs(t) do add(col,v) end; return col end
function add(col,v)
  if v~="?" then
    col.n = col.n + 1
    if not col.isNum then col._has[v] = 1 + (col._has[v] or 0) else
        push(col._has,v)
        col.sorted = false
        col.hi = math.max(col.hi, v)
        col.lo = math.min(col.lo, v)
        if col.n % 2*the.some == 0 then sorted(col) end
        end end end

function sorted(num)
  if not num.sorted then
    table.sort(num._has)
    if #num._has > the.some*1.1 then
      local tmp={}
      for i=1,#num._has,#num._has//the.some do push(tmp,num._has[i]) end
      num._has= tmp end end
  num.sorted = true
  return num. _has end

function div(col)
  if  col.isNum then local a=sorted(col); return (per(a,.9)-per(a,.1))/2.58 else
    local function fun(p) return p*math.log(p,2) end
    local e=0
    for _,n in pairs(_has) do if n>0 then e=e-fun(n/col.n) end end
    return e end end

function mid(col)
  if col.isNum then return per(sorted(col),.5) else
    local most,mode = -1
    for k,v in pairs(_has) do if v>most then most,mode=k,v end end
    return mode end end

---- ---- ---- Data functions
-- Add a new `row` to `data`.
function rowAdd(data,xs)
  xs= push(data.rows, xs.cells and xs or Row(xs))
  for _,todo in pairs{data.cols.x, data.cols.y} do
    for _,col in pairs(todo) do
      add(col, xs.cells[col.at]) end end end

-- Processes table of name strings (from row1 of csv file)
local function _head(sNames)
  local cols = Cols()
  cols.names = namess
  for c,s in pairs(sNames) do
    local col = push(cols.all, -- Numerics start with Uppercase.
                    (s:find"^[A-Z]*" and Num or Sym)(c,s))
    if not s:find":$" then -- some columns are skipped
      push(s:find"[!+-]" and cols.y or cols.x, col) -- some cols are goal cols
      if s:find"!$"    then cols.klass=col end end end
  return cols end

-- if `src` is a string, read rows from file; else read rows from a `src`  table
function load(src)
  local data,fun=Data()
  function fun(t) if data.cols then rowAdd(data,t) else data.cols=_head(t) end end
  if type(src)=="string" then csv(src,fun) else
    for _,t in pairs(src or {}) do fun(t) end end
  return data end

---- ---- ---- ---- Cluster
-- Distance between two rows (returns 0..1)
function dist(data,t1,t2)
  local d = 0
  for _,col in pairs(data.cols.x) do
    local inc = 0
    if   v1=="?" and v2=="?"
    then inc = 1
    else local v1 = norm(col,t1[col.at])
         local v2 = norm(col,t2[col.at])
         if   not col.isNum
         then inc = v1==v2 and 0 or 1
         else if v1=="?" then v1 = v2<.5 and 1 or 0 end
              if v2=="?" then v2 = v1<.5 and 1 or 0 end
              inc = maths.abs(v1-v2) end end
    d = d + inc^the.p
  end
  return (d/data.cols.nx)^(1/the.p) end

-- Numbers get normalized 0..1. Everything esle normalizes to itself.
function norm(col,v)
  if v=="?" or not col.isNum then return v else
    local lo = col.lo[c]
    local hi = col.hi[c]
    return (hi - lo) <1E-9 and 0 or (v-lo)/(hi-lo) end end

return {the=the,mid=mid,div=div,norm=norm,dist=dist,
        Cols=Cols,Num=Num, Sym=Sym, Data=Data}
---- ---- ----  Notes
-- - Each line is usually 80 chars (or less)
-- - Private functions start with `_`
-- - Arguments of private functions do anything at all
-- - Local variables inside functions do anything at all
-- - Arguments of public functions use type hints
--   - Variable  `x` is is anything
--   - Prefix `is` is a boolean
--   - Prefix `fun` is a function
--   - Prefix `f` is a filename
--   - Prefix `n` is a string
--   - Prefix `s` is a string
--   - Prefix `c` is a column index
--   - `col` denotes `num` or `sym`
--   - `x` is anything (table or number of boolean or string
--   - `v` is a simple value (number or boolean  or  string)
--   - Suffix `s` is a list of things
--   - Tables are `t` or, using the above, a table of numbers would be `ns`
--   - Type names are lower case versions of constuctors. so in this code,
--     `cols`,`data`,`num`,`sym` are made by functions `Cols` `Data`, `Num`, `Sym`
```