```lua
                                 __         ___
                               /'__`\      /\_ \
  ____     __      ___ ___    /\ \/\ \     \//\ \    __  __     __
 /',__\  /'__`\  /' __` __`\  \ \ \ \ \      \ \ \  /\ \/\ \  /'__`\
/\__, `\/\ \L\.\_/\ \/\ \/\ \  \ \ \_\ \  __  \_\ \_\ \ \_\ \/\ \L\.\_
\/\____/\ \__/.\_\ \_\ \_\ \_\  \ \____/ /\_\ /\____\\ \____/\ \__/.\_\
 \/___/  \/__/\/_/\/_/\/_/\/_/   \/___/  \/_/ \/____/ \/___/  \/__/\/_/


-- In this code:
-- - Line strive to be 80 chars (or less)
-- - Two spaces before function argumnets denote optionals.
-- - Four spaces before function argumnets denote local variables.
-- - 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`
local l=require"lib0"
local the=l.settings([[
SAM0 : semi-supervised multi-objective explainations
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua eg0.lua [OPTIONS]

OPTIONS:
 -e  --eg     start-up example         = nothing
 -h  --help   show help                = false
 -n  --nums   how many numbers to keep = 256
 -p  --p      distance coeffecient     = 2
 -s  --seed   random number seed       = 10019]])

local copy,csv,o,oo = l.coerce,l.copy,l.csv,l.o,l.oo
local per,push      = l.per, l.push

local adds,add, dist,div,mid,nums,read, record
local Cols, Data, Num, Row, Sym

---- ---- ---- ---- 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, in `cells` (and `cooked` is for discretized data).
function Row(t) return {cells=t, cooked=copy(t)} end

---- ---- ---- ---- Data Functions
---- ---- ---- Update
-- Add one or more items, to `col`. From Num, keep at most `nums` items.
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
       col.lo = math.min(v, col.lo)
       col.hi = math.max(v, col.hi)
       local pos
       if      #col._has < the.nums           then pos = 1 + (#col._has)
       elseif math.random() < the.nums/col.n then pos = math.random(#col._has) end
       if pos then col.sorted = false
                   col._has[pos] = tonumber(v) end end end end

---- ---- ---- Query
-- Return kept numbers, sorted.
function nums(num)
  if not num.sorted then table.sort(num._has); num.sorted=true end
  return num._has end

-- Diversity (standard deviation for Nums, entropy for Syms)
function div(col)
  if  col.isNum then local a=nums(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

-- Central tendancy (median for Nums, mode for Syms)
function mid(col)
  if col.isNum then return per(nums(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
---- ---- ---- Create
-- 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 read(src)
  local data,fun=Data()
  function fun(t) if data.cols then record(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

---- ---- ---- Update
-- Add a new `row` to `data`, updating the `cols` with the new values.
function record(data,xs)
  local row= push(data.rows, xs.cells and xs or Row(xs)) -- ensure xs is a Row
  for _,todo in pairs{data.cols.x, data.cols.y} do
    for _,col in pairs(todo) do
      add(col, row.cells[col.at]) end end end

---- ---- ---- Query
-- For `showCols` (default=`data.cols.x`) in `data`, report `fun` (default=`mid`).
function stats(data,  showCols,fun,    t)
  showCols, fun = showCols or data.cols.y, fun or mid
  t={}; for _,col in pairs(showCols) do t[col.name]=fun(col) end; return t end

---- ---- ---- ---- Distance functions
-- Distance between two values`v1,v2`  within `col`
local function _dist1(col,  v1,v2)
  if   v1=="?" and v2=="?" then return 1 end
  if  not col.isNum         then return v1==v2 and 0 or 1 end
  local function norm(n) return (n-col.lo)/(col.hi-col.lo + 1E-32) end
  if     v1=="?" then v2=norm(v2); v1 = v2<.5 and 1 or 0
  elseif v2=="?" then v1=norm(v1); v2 = v1<.5 and 1 or 0
  else   v1,v2 = norm(v1), norm(v2) end
  return  maths.abs(v1-v2) end

-- Distance between two rows (returns 0..1)
function dist(data,t1,t2)
  local d = 0
  for _,col in pairs(data.cols.x) do
    d = d + _dist1(col, t1.cells[col.at], t2.cells[col.at])^the.p end
  return (d/#data.cols.x)^(1/the.p) end

-- That's all folks.
return {the=the,add=add,adds=adds,mid=mid,div=div,dist=dist,
        nums=nums,record=record,
        Cols=Cols,Num=Num, Sym=Sym, Data=Data}

 _   _   _        __
| | (_) | |__    /  \
| | | | | '_ \  | () |
|_| |_| |_.__/   \__/

local l={}

l.b4={}; for k,v in pairs(_ENV) do l.b4[k]=v end

---- ---- ---- Lists
-- Add `x` to a list. Return `x`.
function l.push(t,x) t[1+#t]=x; return x end

-- Round
function l.rnd(n, nPlaces)
  local mult = 10^(nPlaces or 3)
  return math.floor(n * mult + 0.5) / mult end

-- Deepcopy
function l.copy(t)
  if type(t) ~= "table" then return t end
  local u={}; for k,v in pairs(t) do u[k] = l.copy(v) end
  return u end

-- Return the `p`-th thing from the sorted list `t`.
function l.per(t,p)
  p=math.floor(((p or .5)*#t)+.5); return t[math.max(1,math.min(#t,p))] end

---- ---- ---- Settings
function l.settings(s)
  local t={}
  s:gsub("\n [-][%S]+[%s]+[-][-]([%S]+)[^\n]+= ([%S]+)",
         function(k,x) t[k]=l.coerce(x)end)
  t._help = s
  return t end

function l.cli(t)
  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] = l.coerce(v) end
  if t.help then os.exit(print("\n"..t._help.."\n")) end
  return t end

---- ---- ---- Strings
-- `oo` prints the string from `o`.
-- `o` generates a string from a nested table.
function l.oo(t) print(l.o(t)) return t end
function l.o(t)
  if type(t) ~=  "table" then return tostring(t) end
  local function show(k,v)
    if not tostring(k):find"^_"  then
      v = l.o(v)
      return #t==0 and string.format(":%s %s",k,v) or tostring(v) end end
  local u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
  if #t==0 then table.sort(u) end
  return (t._is or "").."{"..table.concat(u," ").."}" end

-- Convert string to something else.
function l.coerce(s)
  local function coerce1(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 coerce1(s:match"^%s*(.-)%s*$") end

-- Iterator over csv files. Call `fun` for each record in `fname`.
function l.csv(fname,fun)
  local src = io.input(fname)
  while true do
    local s = io.read()
    if not s then return io.close(src) else
      local t={}
      for s1 in s:gmatch("([^,]+)") do t[1+#t] = l.coerce(s1) end
      fun(t) end end end

---- ---- ----
return l
                 __
  ___   __ _    /  \
 / -_) / _` |  | () |
 \___| \__, |   \__/
       |___/
local l=require"lib0"
local _=require"sam0"
local copy,cli = l.copy, l.cli
local o,oo,per,rnd = l.o, l.oo, l.per,l.rnd
local add,div,mid,the  = _.add, _.div,_.mid,_.the
local Num      = _.Num

local eg,fails = {},0
--------------------------------------------------------------------------------
function eg.the() oo(the); return true end

function eg.num(  num)
  num=Num()
  the.nums = 100
  for i=1,100 do add(num,i) end
  print(mid(num) ,rnd(div(num),2))
  return 50==mid(num) and 31.01==rnd(div(num),2)  end

function eg.bignum(  num)
  num=Num()
  the.nums = 32
  for i=1,1000 do add(num,i) end
  oo(_.nums(num))
end

function eg.load() oo(load("../../data/auto93.csv").cols); return true end

local function _egs(   t)
  t={}; for k,_ in pairs(eg) do t[1+#t]=k end; table.sort(t);  return t end

function eg.ls()
  print("\nExamples (lua eg0.lua -f X):\nX=")
  for _,k in pairs(_egs()) do print(string.format(" %-7s",k)) end
  return true end

local function run(k,    b4,out)
  math.randomseed(the.seed)
  b4=copy(the); out=eg[k](); the=copy(b4); return out==true end

function eg.all()
  for _,k in pairs(_egs()) do
    if k ~= "all" then
      if not run(k) then fails = fails + 1; print("FAIL!",k) end end end
  return true end

------------------------------------------------------------------------------
the = cli(the)
if eg[the.eg] then run(the.eg)  end
for k,v in pairs(_ENV) do if not l.b4[k] then print("?",k,type(v)) end end
os.exit(fails)
```