```


   ____    __      ___ ___
  /',__\  /'__`\   /' __` __`\
 /\__, `\/\ \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; e.g `col` isa `Cols`.
local l=require"lib"
local _=require"sam"

local o,oo,per,push,rnd = l.o,l.oo,l.per,l.push,l.rnd
local add,adds,dist,div = _.add,_.adds,_.dist,_.div
local mid, read, the = _.mid,_.read,_.the
local Num,Sym      = _.Num, _.Sym

local eg= {}
function eg.the() oo(the); return true end

function eg.ent(  sym,ent)
  sym= adds(Sym(), {"a","a","a","a","b","b","c"})
  ent= div(sym)
  print(ent,mid(sym))
  return 1.37 <= ent and ent <=1.38 end

function eg.num(  num)
  num=Num()
  for i=1,100 do add(num,i) end
  local med,ent = mid(num), rnd(div(num),2)
  print(mid(num) ,rnd(div(num),2))
  return 50<= med and med<= 52 and 30.5 <ent and ent <32 end

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

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

function eg.dist(  data,t)
  data=read("../../data/auto93.csv")
  t={}
  for i=1,20 do push(t,rnd(dist(data,l.any(data.rows), l.any(data.rows)),2)) end
  table.sort(t)
  oo(t)
  return true end

-- ------------------------------------------------------------------------
the = l.cli(the)
os.exit( l.run(the.eg, eg, the))
```

```

  ___   __ _   _ __
 (_-<  / _` | | '  \
 /__/  \__,_| |_|_|_|

local l=require"lib"
local the=l.settings([[
SAM : Semi-supervised And Multi-objective explainations
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua eg.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]])
-- Commonly used lib functions.
local o,oo,per,push = l.o,l.oo,l.per, l.push

---- ---- ---- ---- Classes
local Data,Cols,Sym,Num,Row
-- 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=l.copy(t)} end

---- ---- ---- ---- Data Functions
local add,adds,clone,div,mid,norm,nums,record,read,stats
---- ---- ---- Update
-- Add one `col`. For Num, keep at most `nums` items.
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

-- Add many items
function adds(col,t) for _,v in pairs(t) do add(col,v) end; return col 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

-- Normalized numbers 0..1. Everything else normalizes to itself.
function norm(col,n)
  return x=="?" or not col.isNum and x or  (n-col.lo)/(col.hi-col.lo + 1E-32) 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(col._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(col._has) do if v>most then mode,most=k,v end end
    return mode end end

-- 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
```

```
---- ---- ---- 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
-- When reading, use row1 to define the column headers.
function read(src,  data,     fun)
  data = data or Data()
  function fun(t) if data.cols then record(data,t) else data.cols=_head(t) end end
  if type(src)=="string" then l.csv(src,fun)
                         else for _,t in pairs(src or {}) do fun(t) end end
  return data end

-- Return a new data with same structure as `data1`. Optionally, oad in `rows`.
function clone(data1,  rows)
  data2=Data()
  data2.cols = _head(data1.cols.names)
  for _,row in pairs(rows or {}) do record(data2,row) end
  return data2 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

---- ---- ---- ---- Distance functions
local dist
-- Distance between two rows (returns 0..1). For unknown values, assume max distan
ce.
function dist(data,t1,t2)
  local function fun(col,  v1,v2)
    if   v1=="?" and v2=="?" then return 1 end
    if not col.isNum then return v1==v2 and 0 or 1 end
    v1,v2 = norm(col,v1), norm(col,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 -------
  local d = 0
  for _,col in pairs(data.cols.x) do
    d = d + fun(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,
        Data=Data, Cols=Cols, Sym=Sym, Num=Num, Row=Row,
        add=add, adds=adds, clone=clone, dist=dist,  div=div,
        mid=mid, nums=nums, read=read, record=record, stats=stats}
```

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

-- lib.lua: misc LUA functions
-- (c)2022 Tim Menzies <timm@ieee.org> BSD-2 licence
local l={}

---- ---- ---- ---- Meta
-- Find rogue locals.
l.b4={}; for k,v in pairs(_ENV) do l.b4[k]=v end
function l.rogues()
  for k,v in pairs(_ENV) do if not l.b4[k] then print("?",k,type(v)) end end end

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

-- Sample one item
function l.any(t) return t[math.random(#t)] end

-- Sample many items
function l.many(t,n,  u)  u={}; for i=1,n do u[1+#u]=l.any(t) end; return u 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 setmetatable(u,getmetatable(t))  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

---- ---- ---- ---- Strings
-- `o` generates a string from a nested table.
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

-- `oo` prints the string from `o`.
function l.oo(t) print(l.o(t)) return t 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

---- ---- ---- ---- Settings
-- Parse help string looking for slot names and default values
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

-- Update `t` from values after command-line flags. Booleans need no values
-- (we just flip the defeaults).
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

---- ---- ---- ---- Main
-- k=`ls`  : list all settings
-- k=`all` : run all demos
-- k=x     : cache settings. reset settings, run one `fun`, update fails counter.

function l.run(k,funs,settings)
  local fails =0
  local function _egs(   t)
    t={}; for k,_ in pairs(funs) do t[1+#t]=k end; table.sort(t); return t end
  if k=="ls" then
    print("\nExamples -e X):\nX=")
    print(string.format(" %-7s","all"))
    print(string.format(" %-7s","ls"))
    for _,k in pairs(_egs()) do print(string.format(" %-7s",k)) end
  elseif k=="all" then
    for _,k in pairs(_egs()) do
      fails=fails + (l.run(k,funs,settings) and 0 or 1) end
  elseif funs[k] then
    math.randomseed(settings.seed)
    local b4={}; for k,v in pairs(settings) do b4[k]=v end
    local out=funs[k]()
    for k,v in pairs(b4) do settings[k]=v end
    print("!!!!!!", k, out and "PASS" or "FAIL") end
  l.rogues()
  return fails end

-- ---------------------------------------------------
-- That's all folks.
return l
```