```lua
  __                                  ____
/\ \__    __                         /\_ \
\ \ ,_\  /\_\     ____   __  __      \//\ \     __  __    __
 \ \ \/  \/\ \  /',__`\ /\ \/\ \       \ \ \   /\ \/\ \  /'__`\
  \ \ \_  \ \ \ /\__, `\\ \ \_\ \  __   \_\ \_ \ \ \_\ \/\ \L\.\_
   \ \__\  \ \_\\/\____/ \/`____ \/\_\  /\____\ \ \____/\ \__/.\_\
    \/__/   \/_/ \/___/   `/___/> \/_/  \/____/  \/___/  \/__/\/_/
                             /\___/
                             \/__/
local _ = require"etc"
local any = _,any
local cat,cli,coerce,copy,csv    = _.cat, _.cli, _.coerce, _.copy, _.csv
local lines,many,obj,push,rogues = _.lines, _.many, _.obj,_.push, _.rogues
local words = _.words

local rand = math.random

local Cols, Data, Num, Sym = obj"Cols", obj"Data", obj"Num", obj"Sym"
local the={ratios-256, bins=8, seed=10019, some=512}

-- Num ---------------------------------------------------------------------------
function Num:new(at,txt)
  txt = txt or ""
  return {n=0,at=at or 0, txt=txt, ready=false, has={},
          hi= -math.huge, lo= math.huge, w=txt:find"-$" and -1 or 1} end

function Num:add(x)
  if x ~= "?" then
    local pos
    self.n  = self.n + 1
    self.lo = math.min(x, self.lo)
    self.hi = math.max(x, self.hi)
    if     #self.has < the.ratios        then pos = 1 + (#self.has)
    elseif rand()    < the.ratios/self.n then pos = rand(#self.has) end
    if pos then self.ready=false
                self.has[pos]=x end end end

function Num:discretize(x)
  local b = (self.hi - self.lo)/the.bins
  return self.hi==self.lo and 1 or math.floor(x/b+.5)*b  end

function Num:dist(x,y)
   if x=="?" and y=="?" then return 1 end
   if     x=="?" then y=self:norm(y); x=y<.5 and 1 or 0
   elseif y=="?" then x=self:norm(x); y=x<.5 and 1 or 0
   else   x,y = self:norm(x), self:norm(y) end
  return math.abs(x-y) end

function Num:holds()
  if not self.ready then table.sort(self.has); self.ready=true end
  return self.has end

function Num:norm(num)
  return self.hi - self.lo < 1E-9 and 0 or (num-self.lo)/(self.hi-self.lo) end

-- Sym ---------------------------------------------------------------------------
function Sym:new(at,txt)
  return {n=0,at=at or 0, txt=txt or "", ready=false, has={}} end

function Sym:add(x)
  if x ~= "?" then
    self.n = self.n + 1
    self.has[x] = 1+(self.has[x] or 0) end end

function Sym:discretize(x) return x end
function Sym:dist(x,y)
    return (x=="?" or y=="?") and 1 or x==y and 0 or 1 end


-- Row ---------------------------------------------------------------------------
function Row:new(cells) return {cells=cells, cooked=copy(cells)} end

-- Cols --------------------------------------------------------------------------
function Cols:new(names)
  self.names, self.x, self.y, self.all= names, {}, {}, {}
  for at,txt in pairs(names) do
    local what = txt:find"^[A-Z]" and Num or Sym
    local col  = push(self.all, what(at,txt))
    if not txt:find":$" then
      push(txt:find"[!+-]$" and self.y or self.x, col) end end end

-- Data --------------------------------------------------------------------------
local function rows(src)
  if type(src) == "table" then return src else
    local u={}; csv(src, function(t) push(u,t) end); return u end end

function Data:new(rows)
  self.rows, self.cols = {},{}
  for i,row in pairs(rows) do
    if   i==1
    then self.cols = Cols(row)
    else push(self.rows, Row(row))
         for cols in pairs{self.cols.x, self.cols.y} do
           for col in pairs(cols) do col:add(row[col.at]) end end end end
  for cols in pairs{self.cols.x, self.cols.y} do
    for _,row in pairs(self.rows) do
        row.cooked[col.at] = col:discretize(row.cells[col.at]) end end end

function Data:dist(row1,row2)
function Data:around(row1, rows)
  return sort(map(rows, function(row2) return {row=row2,d = row1-row2} end),--#
             lt"d") end
function Data.far(XXX) end


function Data:half(rows, above, all)
   local all  = all or self.rows
   local some = many(all, the.some)
   local left = above or far(any(some), some)
-- (defmethod half ((i rows) &optional all above)
--   "Split rows in two by their distance to two remove points."
--   (let* ((all   (or    all (? i _has)))
--          (some  (many  all (! my some)))
--          (left  (or    above (far (any some) some)))
--          (right (far   left some))
--          (c     (dists left right))
--          (n 0)  lefts rights)
--     (labels ((project (row)
--                (let ((a (dists row left))
--                      (b (dists row right)))
--                  (cons (/ (+ (* a a) (* c c) (- (* b b))) (* 2 c)) row))))
--       (dolist (one (sort (mapcar #'project all) #'car<))
--         (if (<= (incf n) (/ (length all) 2))
--           (push (cdr one) lefts)
--           (push (cdr one) rights)))
--       (values left right lefts rights c))))
--
-- ------------------------------------------------------------------------------
the = cli(the)
math.randomseed(the.seed)
rogues()
return {Cols, Data, Num, Sym}
    ,
 _ -+- _.
(/, | (_.

local l={}
-- Cache names -----------------------------------------------------------------
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
function l.rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end

-- Lists -----------------------------------------------------------------------
function l.any(t) return t[math.random(#t)] end

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

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

function l.push(t,x) t[1+#t]=x; return x end

-- Print table -----------------------------------------------------------------
function l.cat(t,        show,u,pub)
  if type(t)~="table" then return tostring(t) end
  function show(k,v)
    if not tostring(k):find"^[A-Z]"  then
      v=l.cat(v)
      return #t==0 and string.format(":%s %s",k,v) or tostring(v) end end
  u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
  table.sort(u)
  return (t._is or "").."{"..table.concat(u," ").."}" end

-- Update slots in `t` from command line ---------------------------------------
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
  return t end

-- Define classes --------------------------------------------------------------
function l.obj(name)
  local function new(k,...)
    local self = setmetatable({},k)
    return setmetatable(k.new(self,...) or self,k) end
  local t={_is = name, __tostring = l.cat}
  t.__index = t
  return setmetatable(t,{__call=new}) end

-- Coerce ----------------------------------------------------------------------
function l.coerce(str)
  local function coerce1(str)
    if str=="true"  then return true end
    if str=="false" then return false end
    return str end
  return tonumber(str) or coerce1(str:match"^%s*(.-)%s*$") end

-- Coerce lines from csv file (fiterling result through `fun`).
function l.csv(filename, fun)
  l.lines(filename, function(t) fun(l.words(t,",",l.coerce)) end) end

--- Call `fun  on all lines from `filename`.
function l.lines(filename, fun)
  local src = io.input(filename)
  while true do
    local str = io.read()
    if not str then return io.close(src) else fun(str) end end end

-- Split  `str` on `sep`, filtering parts through `fun`.
function l.words(str,sep,fun,      t)
  fun = fun or function(z) return z end
  sep = l.string.format("([^%s]+)",sep)
  t={};for x in str:gmatch(sep) do t[1+#t]=fun(x) end;return t end

return l
```