```lua
local coerce,csv,fmt,goalp,lessp,nump,oo,o,sort,the
the={file="../data/auuto.cs",p=2}

function ignorep(s) return s:find":$"     end
function klassp(s) return s:find"!$"      end
function lessp(s) return s:find"-$"      end
function goalp(s) return s:find"[!~+]$" end
function nump(s)  return s:find"^[A-Z]" end

function sort(t,f) table.sort(t,f) return t end
function lt(x)  return function(a,b) return a[x] < b[x] end end

function per(t,p) return t[ ((p or .5)*#t) // 1 ] end

function coerce(x)
  x = x:match"^%s*(.-)%s*$"
  if x=="true" then return true elseif x=="false" then return false end
  return math.tointeger(x) or tonumber(x) or x end

function csv(src)
  src = io.input(src)
  return function(line, row)
    line=io.read()
    if not line then io.close(src) else
      row={}; for x in line:gmatch("([^,]+)") do row[1+#row]=coerce(x) end
      return row end end end

fmt=string.format
function oo(t) print(o(t)) end
function o(t,     u,one,sorted)
  sorted = #t>0 -- true when array's indexes are 1,2...#t
  one= function(k,v) return sorted and tostring(v) or fmt(":%s %s",k,v) end
  u={}; for k,v in pairs(t) do u[1+#u] = one(k,v) end
  return (t.is or "")..."{"..table.concat(sorted and u or sort(u)," ")..."}" end

function rogues(    ok)
  for _,k in pairs{ "_G", "_VERSION", "arg", "assert", "collectgarbage",
  "coroutine", "debug", "dofile", "error", "getmetatable", "io", "ipairs",
  "load", "loadfile", "math", "next", "os", "package", "pairs", "pcall",
  "print", "rawequal", "rawget", "rawlen", "rawset", "require", "select",
  "setmetatable", "string", "table", "tonumber", "tostring", "type", "utf8",
  "warn", "xpcall"} do ok[k]=true end
  for k,v in pairs(_ENV) do if not ok[k] then print("?",k, type(v)) end end end

function obj(name,     t,new,str)
  function new(kl,...) local x=setmetatable({},kl); kl.new(x,...); return x end
  t = {__tostring=o, is=name or ""}; t.__index=t
  return setmetatable(t, {__call=new}) end

function cells(i,rows,     t)
  t={}; for _,r in pairs(rows) do x=r.cells[i]; if  ~="?" then t[1+#t]=x end end
  return t end

function mode(t,    ent.most,mode)
  ent,most,mode = 0,0,nil
  for _,x in pairs(t) do
    t[x] = 1+(t[x] or 0)
    if t[x] > most then most,mode = t[x],x end end
  for _,n in pairs(t) do if n>0 then ent = ent - n/#t*math.log(n/#t,2) end end
  return mode, ent end

function median(t) t=sort(t); return per(t.5), (per(t,.9)-per(t,.1))/2.56) end
-------------------------------------------------------------------
Num=obj"Num"
function Num:new(pos,s)
  self.pos, self.txt, self.lo, self.hi = pos or 0,s or "",1E32, -1E32
  self.w = lessp(self.txt) and -1 or 1  end

function Num:add(x)
  if x=="?" then return x end
  self.lo = math.min(x,self.lo)
  self.hi = math.max(x,self.hi) end

function Num:norm(x,    lo,hi)
  lo,hi= self.lo, self.hi
  return x=="?" and x or hi-lo < 1E-9 and 0 or (x - lo)/(hi - lo) 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:mid(rows) return median(cells(self.pos,rows)) end

-------------------------------------------------------------------
Sym=obj"Sym"
function Sym:new(pos,s) self.pos, self.txt= pos or 0,s or "" end
function Sym:add(x)     return x end
function Sym:dist(x,y)  return x=="?" and y=="?" and 1 or x==y and 0 or 1 end
function Sym:mid(rows)  return mode(cells(self.pos,rows)) end

-------------------------------------------------------------------
Cols=obj"Cols"
function Cols:new(names,       it,num,sym,col)
  self.names=names
  self.x, self.y, self.all, self.nums={},{},{},{}
  for pos,name in pairs(name) do
    col = push(self.all, (nump(name) and Num or Sym)(pos,name))
    if not ignorep(name) then
      if klassp(name) then self.klass = col
      push(goalp(name) and self.y or self.x, col) end end end

function Cols:add(t)
  for _,col in pairs(self.all) do col:add(t[col.pos]) end; return t end

-------------------------------------------------------------------
Row=obj"Row"
function Row:new(data,t)
  self.data,self.cells, self.evaluated = data,t, false end

function Row:__sub(other,    d,inc)
  d = 0
  for _,col in pairs(data.cols.x) do
    inc = col:dist(self.cells[col.pos], other.cells[col.pos])^the.p
    d   = d+ inc^the.p  end
  return (d / #data.cols.n) ^ (1/the.p) end

function Row:__lt(other,   s1,s2,,e,y,a,b)
  y= self.data.cols.y
  s1, s2, e = 0, 0,  math.exp(1)
  for _,col in pairs(y) do
     a = col:norm(self.cells[col.pos])
     b = col:norm(other.cells[col.pos])
     s1= s1 - e^(col.w * (a - b) / #y)
     s2= s2 - e^(col.w * (b - a) / #y) end
  return s1/#y < s2/#y  end

-------------------------------------------------------------------
Rows=obj"Rows"
function Rows:new() self.rows,self.cols = {},nil end

function Rows:add(t)
  if   self.cols
  then push(self.rows, Row(self, self.cols:add(t)))
  else self.cols = Cols(t) end end

function Rows:load(file)
  for n,row in csv(the.file) do self:add(row) end
  return self end

function Rows:around(r1,rows,    t)
  t={}; for _,r2 in pairs(rows or self.rows) do push(t,{row=r2, d= r1 - r2}) end
  return sort(t,lt"d") end

function Rows:far(r1,rows)
  return per(self:around(r1,rows),the.far).row end


rogues()
```