```lua
#!/usr/bin/env lua
-- vim : ft=lua :
local your = {} -- user settings (may be changes from command-line)
local our  = {} -- system settings (controlled internal to code)
our.help   = [[

./keys0 [OPTIONS]
(c)2022, Tim Menzies <timm@ieee.org>, unlicense.org

  -better true
  -Debug  true
  -far    .9
  -file   ../../data/auto93.csv
  -h      false
  -goal   smile
  -p      2
  -round  2
  -seed   10019
  -Some   512
  -todo   all]]

our.b4={}        -- globals known, pre-code. used to find stray globals
for k,_ in pairs(_ENV) do our.b4[k]=k end

local add, any, asserts,coerce, col, copy, csv, defaults, dist
local fmt, klass, map, main, new,o, push, rand, randi, rnd, rnds
local same, slots, sort, xpect

function klass(s, it)
  it = {_is=s, __tostring=o}
  it.__index = it
  return setmetatable(it,{__call=function(_,...) return it.new(...) end}) end

local COLS,EG,EGS   = klass"COLS", klass"EG", klass"EGS"
local NUM,RANGE,SYM = klass"NUM", klass"RANGE", klass"SYM"

------------------------------------------------------------------------------
function NUM.new(at,s,   i)
  i = new(NUM,{n=0, at=at or 0, txt=s or "", mu=0, m2=0,
               lo=math.huge, hi=-math.huge})
  i.w = i.txt:find"-" and -1 or 1
  return i end

function NUM.add(i,x,  d)
  if x~="?" then
    i.n = i.n + 1
    d    = x    - i.mu
    i.mu = i.mu + d/i.n
    i.m2 = i.m2 + d*(x-i.mu)
    i.lo = math.min(i.lo,x); i.hi = math.max(i.hi,x) end
  return x end

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

function NUM.div(i) return i.n<2 and 0 or (i.m2/(i.n-1))^0.5 end

function NUM.mid(i) return i.mu end

function NUM.norm(i,x) return i.hi-i.lo<1E-9 and 0 or (x-i.lo)/(i.hi-i.lo) end

------------------------------------------------------------------------------
function SYM.new(at,s)
  return new(SYM,{n=0, at=at or 0, txt=s or "", has={}, most=0, mode=nil}) end

function SYM.add(i,x,count)
  count = count or 1
  i.has[x] = count + (i.has[x] or 0)
  if i.has[x] > i.most then i.most,i.mode = i.has[x], x end
  return x end

function SYM.dist(i,x,y) return x=="?" and y=="?" and 1 or x==y and 0 or 1 end

function SYM.div(i,   e)
  e=0; for _,n in pairs(i.has) do e=e-n/i.n*math.log(n/i.n,2) end; return e end

function SYM.merged(i,j,     k)
  k= SYM(i.at, i.txt)
  for x,count in pairs(i.has) do k:add(x,count) end
  for x,count in pairs(j.has) do k:add(x,count) end
  return k end

function SYM.mid(i) return i.mode end

------------------------------------------------------------------------------
function EG.new(t) return new(EG, {cooked={}, has=t}) end

function EG.better(eg1,eg2,egs)
  local s1,s2,e,n,a,b = 0,0,10,#egs.cols.y
  for _,col in pairs(egs.cols.y) do
    a  = col:norm(eg1.has[col.at])
    b  = col:norm(eg2.has[col.at])
    s1 = s1 - e^(col.w * (a-b)/n)
    s2 = s2 - e^(col.w * (b-a)/n) end
  return s1/n < s2/n end

function EG.cols(i,cols) return map(cols,function(x) return i.has[x.at] end) end

function EG.dist(i,j,egs,    a,b,d,n)
  d,n = 0, #egs.cols.x + 1E-31
  for _,col in pairs(egs.cols.x) do
    a,b = i.has[col.at], j.has[col.at]
    d   = d + col:dist(a,b) ^ your.p end
  return (d/n) ^ (1/your.p) end


------------------------------------------------------------------------------
function RANGE.new(col,lo,hi,has)
  lo = lo or -math.huge
  return new(RANGE, {score=nil,col=col, lo=lo, hi=hi or lo, has=has or SYM()}) e
nd

function RANGE.__tostring(i)
  if i.lo == i.hi       then return fmt("%s == %s",i.col.txt,i.lo) end
  if i.lo == -math.huge then return fmt("%s < %s",i.col.txt,i.hi) end
  if i.ho ==  math.huge then return fmt("%s >= %s",i.col.txt,i.lo) end
  return fmt("%s <= %s < %s", i.col.txt, i.lo, i.hi) end

function RANGE.select(i,eg,       x)
  x = eg.has[i.col.at]
  return x=="?" or i.lo <= x and x < i.hi end

function RANGE.merge(i,j,      k)
 k = RANGE(i.col, i.lo, j.hi, i.has:merged(j.has))
 if k.has:div()*1.01 <= xpect(i.has, j.has) then return k end end

function RANGE.eval(i,goal)
    local best, rest, goals = 0,0,{}
  if not i.score then
    function goals.smile(b,r) return r>b and 0 or b*b/(b+r +1E-31) end
    function goals.frown(b,r) return b<r and 0 or r*r/(b+r +1E-31) end
    function goals.xplor(b,r) return 1/(b+r                +1E-31) end
    function goals.doubt(b,r) return 1/(math.abs(b-r)      +1E-31) end
    for x,n in pairs(i.has) do
      if x==goal then best = best+n/i.n else rest = rest+n/i.n end end
    i.score = best + rest < 0.01 and 0 or goals[your.goal](best,rest) end
  return i.score end

------------------------------------------------------------------------------
function COLS.new(eg,     i,now,where)
  i = new(COLS,{all={}, x={}, y={}})
  for at,s in pairs(eg) do    -- First row. Create the right columns
    now   = push(i.all, (s:find"^[A-Z]" and NUM or SYM)(at,s))
    where = (s:find"-" or s:find"+") and i.y or i.x
    if not s:find":" then push(where, now) end end
  return i end

function COLS.add(i,eg)
  return map(i.all, function(col) return col:add(eg[col.at]) end) end

------------------------------------------------------------------------------
function EGS.new(i) return new(EGS, {rows={}, cols=nil}) end

function EGS.add(i,eg)
  eg = eg.has and eg.has or eg -- If eg has data buried inside, expose it.
  if i.cols then push(i.rows,EG(i.cols:add(eg))) else i.cols=COLS(eg) end end

function EGS.clone(i,inits,    j)
  j = EGS()
  j:add(map(i.cols.all, function(col) return col.txt end))
  for _,x in pairs(inits or {}) do  j:add(x) end
  return j end

function EGS.cluster(i, top)
  local zero,one,two,ones,twos,both,a,b,c
  top   = top or i
  zero  = any(i.rows)
  one   = top:far(zero)
  two,c = top:far(one)
  ones,twos,both = i:clone(), i:clone(),{}
  for _,eg in pairs(i.rows) do
    a = eg:dist(one, top)
    b = eg:dist(two, top)
    push(both, {(a^2 + c^2 - b^2) / (2*c),eg}) end
  for n,pair in pairs(sort(both, function(a,b) return a[1] < b[1] end)) do
    (n <= #both//2 and ones or twos):add(pair[2]) end
  if your.better and two:better(one,i) then ones,twos=twos,ones end
  return ones, twos end

function EGS.far(i,eg1,    fun,tmp)
  fun = function(eg2) return {eg2, eg1:dist(eg2,i)} end
  tmp = #i.rows > your.Some and any(i.rows, your.Some) or i.rows
  tmp = sort(map(tmp, fun), function(a,b) return a[2] < b[2] end)
  return table.unpack(tmp[#tmp*your.far//1] ) end

function EGS.from(t, i)
  i=i or EGS(); for _,eg in pairs(t) do i:add(eg) end; return i end

function EGS.mid(i,cols)
  return map(cols or i.all, function(col) return col:mid() end) end

function EGS.read(file, i)
  i=i or EGS(); for eg in csv(file) do i:add(eg) end; return i end
```

```lua
----------------------------------------------------------------------------
function any(t,  n)
  if not n then return t[randi(1,#t)] end
  u={};for j=1,n do push(u,any(t)) end; return u end

our.fails = 0
function asserts(test,msg)
  msg=msg or ""
  if test then return print(" PASS : "..msg) end
  our.fails = our.fails+1
  print(" FAIL : "..msg)
  if your.Debug then assert(test,msg) end end

function coerce(x)
  if x=="true" then return true elseif x=="false" then return false end
  return tonumber(x) or x end

function copy(t,u)
  u={}; for k,v in pairs(t) do u[k]=v end
  return setmetatable(u, getmetatable(t)) end

function csv(file,   x,row)
  function row(x,  t)
    for y in x:gsub("%s+",""):gmatch"([^,]+)" do push(t,coerce(y)) end
    return t
  end -----------------
  file = io.input(file)
  return function()
    x=io.read(); if x then return row(x,{}) else io.close(file) end end end

function defaults(help_string,       t,fun)
  function fun(flag,x)
    for n,txt in ipairs(arg) do
      if   txt:sub(1,1)=="-" and flag:match("^"..txt:sub(2)..".*")
      then x = x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
    t[flag] = coerce(x)
  end -----------------
  t = {}
  help_string:gsub("\n [-]([^%s]+)[^\n]*%s([^%s]+)", fun)
  return t end

function fmt(...) return string.format(...) end

function main(our,your)
  our.defaults = defaults(our.help)
  for k,v in pairs(our.defaults) do your[k] = v end
  if your.h then os.exit(print(our.help)) end
  our.fails = 0
  for _,one in pairs(your.todo=="all" and slots(our.go) or {your.todo}) do
    for k,v in pairs(our.defaults) do your[k] = v end
    our.go[one]()
  end -----------
  for k,v in pairs(_ENV) do
    if not our.b4[k] then print("?rogues",k,type(v)) end end
  return our.fails end

function map(t,f,  u)
  u= {};for k,v in pairs(t) do push(u,(f or same)(v)) end; return u end

our.oid=0
function new(mt,x)
  our.oid = our.oid+1; x._oid = our.oid -- Everyone gets a unique id.
  return setmetatable(x,mt) end        -- Methods now delegate to `mt`.

function o(t)
  local u,key
  key= function(k) return fmt(":%s %s", k, o(t[k])) end
  if type(t) ~= "table" then return tostring(t) end
  u = #t>0 and map(t,o) or map(slots(t),key)
  return (t._is or "").."{"..table.concat(u, " ").."}" end

function push(t,x) table.insert(t,x); return x end

your.seed = your.seed or 10019
function rand(lo,hi)
  your.seed = (16807 * your.seed) % 2147483647
  return (lo or 0) + ((hi or 1) - (lo or 0)) * your.seed / 2147483647 end

function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end

function rnd(x,d,  n)
  if type(x)~="number" then return x end
  n=10^(d or your.round)
  return math.floor(x*n+0.5)/n end

function rnds(t,d) return map(t,function(x) return rnd(x,d) end) end

function same(x,...) return x end

function slots(t,   u)
  u={}
  for k,_ in pairs(t) do if tostring(k):sub(1,1) ~= "_" then push(u,k) end end
  return sort(u) end

function sort(t,f) table.sort(t,f);   return t end

function xpect(i,j) return (i.n*i:div() + j.n*j:div()) / (i.n + j.n) end


----------------------------------------------------------------------------
our.go={}   -- list of enabled tests
our.nogo={} -- list of disabled test
local go, nogo = our.go,our.nogo

function go.settings()
  print("our",o(our))
  print("your",o(your)) end

function go.range(  r)
  r=RANGE(NUM(10,"fred"),"apple")
  assert(tostring(r) == "fred == apple", "print ok") end

function go.num(    m,n)
  m=NUM();   for j=1,10 do m:add(j) end
  n=copy(m); for j=1,10 do n:add(j) end
  asserts(2.95 == rnd(n:div()),"sd ok") end

function go.egs(    egs)
  egs = EGS.read(your.file)
  asserts(egs.cols.y[1].hi==5140,"most seen") end

function go.clone(     egs1,egs2,s1,s2)
  egs1 = EGS.read(your.file)
  s1   = o(egs1.cols.y)
  egs2 = egs1:clone(egs1.rows)
  s2   = o(egs2.cols.y)
  asserts(s1==s2, "cloning works") end

function go.dist()
  local egs,eg1,dist,tmp,j1,j2,d1,d2,d3,one
  egs  = EGS.read(your.file)
  eg1  = egs.rows[1]
  dist = function(eg2) return {eg2,eg1:dist(eg2,egs)} end
  tmp  = sort(map(egs.rows, dist), function(a,b) return a[2] < b[2] end)
  one  = tmp[1][1]
  for j=1,10 do
    j1 = randi(1,#tmp)
    j2 = randi(1,#tmp)
    if j1>j2 then j1,j2=j2,j1 end
    d1 = tmp[j1][1]:dist(one,egs)
    d2 = tmp[j2][1]:dist(one,egs)
    asserts(d1 <= d2,"distance ") end end

function go.cluster(   top,left,right)
  top = EGS.read(your.file)
  left, right = top:cluster()
  for n,t in pairs{top,left,right} do print(n,o(rnds(t:mid(t.cols.y)))) end
end

os.exit( main(our, your))
```