```lua
#!/usr/bin/env lua
--       _
--    __| |_   _  ___
--   / _` | | | |/ _ \
--  | (_| | |_| | (_) |
--   \__,_|\__,_|\___/  .lua
--
local F=require"fun"
local the=F.options[[

./duo.lua [OPTIONS]
(c)2022 Tim Menzies, MIT license

Data miners using/used by optimizers.
Understand N items after log(N) probes, or less.

OPTIONS:
  -ample   when enough is enough         =  512
  -Debug   on error, dump stack and halt = false
  -enough  use (#t)^enough               =  .5
  -far     how far to go                 =  .9
  -file    read data from file           =  ../etc/data/auto93.csv
  -help    show help                     =  false
  -p       distance coefficient          =   2
  -rnd     default round                 =  %5.2f
  -seed    random number seed            =  10019
  -task    start up actions              =  donothing]]

local EGS, NUM, RANGE, SYM = {},{},{},{}
local any,    asserts,  brange,  firsts,  fmt,  many,  map =
      F.any,F.asserts,F.brange,F.firsts,F.fmt.F.many,F.map
local   new,  o,   oo,  push,  rows,  seconds,  sort =
      F.new,F.oo,F.oo,F.push,F.rows,F.seconds,F.sort
--- ## RANGE
function RANGE.new(k,col,lo,hi,b,B,r,R)
  return new(k,{col=col,lo=lo,hi=hi or lo,b=b,B=B,r=r,R=R}) end

function RANGE.__lt(i,j) return i:val() < j:val() end
function RANGE.merge(i,j,k,   lo,hi)
  lo = math.min(i.lo, j.lo)
  hi = math.max(i.hi, j.lhi)
  k = RANGE:new(i.col,lo,hi,i.b+j.b,i.B,i.r+j.r, j.R)
  if k:val() > i:val() and j:val() then return k end end

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.hi ==  math.huge then return fmt("%s >= %s", i.col.txt, i.lo) end
  return fmt("%s <= %s < %s", i.lo, i.col.txt, i.hi) end

function RANGE.val(i,   z,B,R)
  z=1E-31; B,R = i.B+z, i.R+z; return (i.b/B)^2/( i.b/B + i.r/R) end

function RANGE.selects(i,row,    x)
  x=row.has[col.at]; return x=="?" or i.lo<=x and x<i.hi end
--- ## NUM
function NUM.new(k,at,s)
  return new(k,{at=at,txt=s,w=s:find"-" and -1 or 1,_has={},
                    ok=false, lo=math.huge, hi=-math.huge}) end

function NUM.add(i,x)
  if x ~= "?" then
    i.ok = false
    push(i._has, x)
    if x < i.lo then i.lo = x end
    if x > i.hi then i.hi = x end end
  return x end

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

function NUM.has(i)
  if not i.ok then sort(i._has); i.ok=true end; return i._has end

function NUM.mid(i,   a) a=i:has(); return a[#a//2] end

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

-- compare to old above
-- function NUM.ranges(i,j,lo,hi)
--   local z,is,js,lo,hi,m0,m1,m2,n0,n1,n2,step,most,best,r1,r2
--   is,js    = i:has(), j:has()
--   lo       = math.min(is[1],   js[1])
--   hi       = math.max(is[#is], js[#js])
--   gap, max = (hi - lo)/16, -1
--   for x=lo,hi,gap do
--     --     col, lo hi, b     B   r         R
--     local b =
--     RANGE:new(i,lo,hi,
--   if hi-lo < 2*gap then
--     z       = 1E-32
--     m0, m2 = fun.search(is, lo),fun.bsearch(is, hi+z)
--     n0, n2 =fun.bsearch(js, lo),fun.bsearch(js, hi+z)
--     --                    col,lo hi,b     B   r     R
--     best     = nil
--     for mid in lo,hi,gap do
--       if mid > lo and k < hi then
--         m1 = bsearch(is, mid+z)
--         n1 = bsearch(js, mid+z)
--         r1 = RANGE:new(i,    lo,mid,m1-m0,i.n,m2-(m1+1),j.n)
--         r2 = RANGE:new(i, mid+z,hi, n1-n0,i.n,n2-(n1+1),j.n)
--         if r1:val() > max then best, max = r1, r1:val() end
--         if r2:val() > max then best, max = r2, r2:val() end end end end
--   if   best
--   then return i:ranges(j, best.lo, best.hi)
--   else return RANGE:new(i,  lo,hi,m2-m0,i.n,n2-n0,j.n) end end
--
--- ## SYM
function SYM.new(k,at,s) return new(k,{at=at,txt=s,_has={},mode=nil,most=0}) end
function SYM.add(i,x)
  if x~="?" then
     i._has[x]=1+(i._has[x] or 0)
     if i._has[x] > i.most then i.most, i.mode = i._has[x],x end
  end
  return x end

function SYM.dist(i,a,b) return  a=="?" and b=="?" and 1 or a==b and 0 or 1 end
function SYM.has(i)      return i.has end
function SYM.mid(i)      return i.mode end
function SYM.ranges(i,j)
  return lib.mapp(i._has,      -- col lohib B   r                 R
       function(x,n) return RANGE:new(i,x,x,n,i.n,(j._has[x] or 0),j.n) end) end
--- ## EGS
function EGS.new(k,file,   i)
  i= new(k,{_rows={}, cols=nil, x={},  y={}})
  if file then for row in rows(file) do i:add(row) end end
  return i end

function EGS.add(i,t)
  local add,now,where = function(col) return col:add(t[col.at]) end
  if i.cols then
    push(i._rows, map(i.cols, add))
  else
    i.cols = {}
    for n,x in pairs(t) do
      now = push(i.cols, (x:find"^[A-Z]" and NUM or SYM):new(n,x))
      if not x:find":" then
        push((x:find"+" or x:find"-") and i.y or i.x, now) end end end end

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

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

function EGS.dist(i,r1,r2)
  local d,n,inc = 0, (#i.x)+1E-31
  for _,col in pairs(i.x) do
    inc = col:dist(r1[col.at], r2[col.at])
    d   = d + inc^the.p end
  return (d/n)^(1/the.p) end

function EGS.far(i,r1,rows,        act,tmp)
  act = function(r2) return {r2, i:dist(r1,r2)} end
  tmp = sort(map(rows,act), seconds)
  return table.unpack(tmp[#tmp*the.far//1] ) end

function EGS.half(i,rows)
  local some,left,right,c,cosine,lefts,rights
  rows    = rows or i._rows
  some    = #rows > the.ample and many(rows, the.ample) or rows
  left    = i:far(any(rows), some)
  right,c = i:far(left,      some)
  function cosine(r,     a,b)
    a, b = i:dist(r,left), i:dist(r,right); return {(a^2+c^2-b^2)/(2*c),r} end
  lefts,rights = i:clone(), i:clone()
  for n,pair in pairs(sort(map(rows,cosine), firsts)) do
    (n <= (#rows)/2 and lefts or rights):add( pair[2] ) end
  return lefts,rights,left,right,c end

local rnd,show
function EGS.cluster(i, top)
  local c,lefts0, rights0, lefts, rights, left, right=0
  top = top or i
  if #i._rows >=  2*(#top._rows)^the.enough then
    lefts0, rights0, left, right, c = top:half(i._rows)
    lefts  = lefts0:cluster( top)
    rights = rights0:cluster(top)
  end
  return {here=i, lefts=lefts, rights=rights, left=left, right=right, c=c} end

function rnd(x)
  return fmt(type(x)=="number" and x~=x//1 and the.rnd or"%s",x) end

function show(t,lvl)
  lvl = lvl or ""
  if t then
    --if t.lefts
    print(fmt("%s%s",lvl,#t.here._rows))
    --else print(fmt("%s%s\t%s", lvl,#t.here._rows, t.here:mid())) end
    show(t.lefts, lvl.."|.. ")
    show(t.rights,lvl.."|.. ") end end

--- ## Tests and Demo
local no,go={},{}

function go.cluster()  show(EGS:new(the.file):cluster())  end

function go.half(  a,b)
  local lefts,rights,left,right,c=EGS:new(the.file):half()
  print("rows",#lefts._rows, #rights._rows)
  oo{left=left}
  oo{right=right}
  oo{c=c}
  end

function go.any(   t,x,n)
  t={}; for i=1,10 do t[1+#t] = i end
  n=0; for i=1,5000 do x=any(t); n= 1 <= x and x <=10 and n+1 or 0 end
  asserts(n==5000,"any")  end

function go.bsearch(   t,x,a,b)
  t={}
  for j =1,10^6 do push(t,100*math.random()//1) end
  table.sort(t);
  for j =1,1000 do
     x=any(t)
     a,b = brange(t,x)
     assert(t[a-1] ~= x)
     assert(t[b+1] ~= x)  ----
     for k=a,b do assert(t[k] == x) end end end

function no.fail()       asserts(fail,"checking crashes"); print(no.thi.ng) end
function go.oo(  u)      oo{10,20,30} end
function go.rows( t)
  for row in rows(the.file) do t=row  end
  asserts(type(t[1])=="number","is number")
  asserts(t[1]==4, "is four")
  asserts(#t==8,"is eight") end

function go.egs(   i,t)
  i=EGS:new(the.file); map(i.y,oo)
  print(10)
  asserts(i.y[1].lo==1613,"lo")
  t=i.y[1]:has(); asserts(1613==t[1],"lo2") asserts(5140== t[#t],"hi");
  asserts(i.y[1].ok,"ok") end

function go.dist(  i, t,a,b,d)
  i=EGS:new(the.file)
  t= i._rows
  for j=1,100 do
     a,b= any(t), any(t)
     d= i:dist(a,b)
     assert(0<= d and d <= 1) end end

the(go)
```