```lua
#!/usr/bin/env lua
--   __
--  /\ \
--  \_\ \      __  __        ___
--  /'_` \    /\ \/\ \      / __`\
-- /\ \L\ \   \ \ \_\ \    /\ \L\ \
-- \ \___,_\   \ \____/    \ \____/
--  \/__,_ /    \/___/      \/___/

local your, our={}, {b4={}, help=[[
duo.lua [OPTIONS]
(c)2022 Tim Menzies, MIT license (2 clause)
Data miners using/used by optimizers.
Understand N items after log(N) probes, or less.

  -file   ../../data/auto93.csv
  -ample  512
  -far   .9
  -best  .5
  -help  false
  -dull  .5
  -rest  3
  -seed  10019
  -Small .35
  -rnd   %.2f
  -task  -
  -p     2]]}

for k,_ in pairs(_ENV) do our.b4[k] = k end
local any,asserts,cells,copy,first,firsts,fmt,go,id,main,many,map
local merge,new,o,push,rand,randi,ranges,rnd,rogues,rows,same
local second, seconds,settings,slots,sort,super,thing,things,xpect
local COLS,EG,EGS,NUM,RANGE,SAMPLE,SYM
local class= function(t,  new)
  function new(_,...) return t.new(...) end
  t.__index=t
  return setmetatable(t,{__call=new}) end

-- Copyright (c) 2022, Tim Menzies
--
-- Redistribution and use in source and binary forms, with or without
-- modification, are permitted provided that the following conditions are met.
-- (1) Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.  (2) Redistributions
-- in binary form must reproduce the above copyright notice, this list of
-- conditions and the following disclaimer in the documentation and/or other
-- materials provided with the distribution.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
-- IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHNTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
-- CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
-- EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
-- PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
-- PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
-- LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
-- NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
-- SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

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

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

function COLS.__tostring(i, txt)
  function txt(c) return c.txt end
  return fmt("COLS{:all %s\n\t:x %s\n\t:y %s", o(i.all,txt), o(i.x,txt), o(i.y,txt)) end

function COLS.add(i,t,      add)
  function add(col,    x) x=t[col.at]; col:add(x);return x end
  return map(i.all, add) end
-- -----------------------------------------------------------------------------
EG=class{}
function EG.new(t) return new({has=t, id=id()},EG) end

function EG.__tostring(i) return fmt("EG%s%s %s", i.id,o(i.has),#i.has) end

function EG.better(i,j,cols)
  local s1,s2,e,n,a,b = 0,0,10,#cols
  for _,col in pairs(cols) do
    a  = col:norm(i.has[col.at])
    b  = col:norm(j.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.col(i,cols)
  return map(cols, function(col) return i.has[col.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
-- -----------------------------------------------------------------------------
EGS=class{}
function EGS.new() return new({rows={}, cols=nil}, EGS) end

function EGS.__tostring(i) return fmt("EGS{#rows %s:cols %s", #i.rows,i.cols) end

function EGS.add(i,row)
  row = row.has and row.has or row
  if i.cols then push(i.rows,EG(i.cols:add(row))) else i.cols=COLS(row) 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.far(i,eg1,rows,    fun,tmp)
  fun = function(eg2) return {eg2, eg1:dist(eg2,i)} end
  tmp = sort(map(rows, fun), seconds)
  return table.unpack(tmp[#tmp*your.far//1] ) end

function EGS.file(i,file) for row in rows(file) do i:add(row) end; return i end

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

function EGS.halve(i,rows)
  local c,l,r,ls,rs,cosine,some
  function cosine(row,     a,b)
    a,b = row:dist(l,i), row:dist(r,i); return {(a^2+c^2-b^2)/(2*c),row} end
  rows  = rows or i.rows
  some  = #rows > your.ample and many(rows, your.ample) or rows
  l     = i:far(any(rows), some)
  r,c   = i:far(l,         some)
  ls,rs = i:clone(), i:clone()
  for n,pair in pairs(sort(map(rows,cosine), firsts)) do
    (n <= #rows//2 and ls or rs):add(pair[2]) end
  return ls,rs,l,r,c end

-- XXX ranges2 suspicious. d=0 and morerangesis 0
function EGS.ranges(i,j,     all,there, ranges)
  all = {}
  for n,here in pairs(i.cols.x) do
    there = j.cols.x[n]
    ranges = here:ranges(there)
    if #ranges> 1 then push(all, {xpect(ranges,here.txt .. "ranges"),ranges}) end

    end
  --for k,v  in pairs(sort(all,firsts)) do
    -- print(v[1], #v[2], v[2][1].col.txt) end
  return map(sort(all,firsts),second) end

function EGS.xcluster(i,top,lvl)
  local split, left, right,kid1, kid2
  top, lvl = top or i, lvl or 0
  ls,rs = (top or i):halve(i.rows)
  if #i.rows >= 2*(#top.rows)^your.small then
    split, kid1, kid2 = i:splitter(top), i:clone(), i:clone()
    for _,row in pairs(i.rows) do
      (split:selects(row) and kid1 or kid2):add(row) end
    if #kid1.rows ~= #i.rows then left  = kid1:xcluster(top,lvl+1) end
    if #kid2.rows ~= #i.rows then right = kid2:xcluster(top,lvl+1) end
  end
  return {here=i, split=split, left=left, right=right} end
-- ------------------------------------------------------------------------------
NUM=class{}
function NUM.new(at,s, big)
  big = math.huge
  return new({lo=big, hi=-big, at=at or 0, txt=s or "",
              n=0, mu=0, m2=0, sd=0,_all=SAMPLE(),
              w=(s or ""):find"-" and -1 or 1},NUM) end

function NUM.__tostring(i)
  return fmt("NUM{:at %s :txt %s :n %s :lo %s :hi %s :mu %s :sd %s}",
             i.at, i.txt,  i.n, i.lo, i.hi, rnd(i.mu), rnd(i:div())) end

function NUM.add(i,x,     d,pos)
  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(x,i.lo); i.hi = math.max(x,i.hi)
    i._all:add(x) 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.div(i) return i.n <2 and 0 or (i.m2/(i.n-1))^0.5 end

function NUM.merge(i,j,  k)
  k= NUM(i.at, i.txt)
  for _,x in pairs(i._all,it) do k:add(x) end
  for _,x in pairs(j._all.it) do k:add(x) end
  return k 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 NUM.ranges(i,j,ykind,       tmp,xys)
  xys={}
```

```
  for _,x in pairs(i._all.it) do push(xys,{x=x,y="best"}) end
  for _,x in pairs(j._all.it) do push(xys,{x=x,y="rest"}) end
  return merge( ranges(xys,i,  ykind or SYM,
                               (#xys)^your.dull,
                               xpect{i,j}*your.Small)) end
-- ------------------------------------------------------------------------------
RANGE=class{}
function RANGE.new(col,lo,hi,ys)
  return new({n=0, col=col, lo=lo, hi=hi or lo, ys=ys or SYM()},RANGE) end

function RANGE.__lt(i,j) return i:div() < j:div() 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.add(i,x,y,inc)
  inc  = inc or 1
  i.n  = i.n + inc
  i.hi = math.max(x,i.hi)
  i.ys:add(y, inc) end

function RANGE.div(i) return i.ys:div() end

function RANGE.selects(i,row,    x)
  x=row.has[col.at]; return x=="?" or i.lo<=x and x<i.hi end
-- ------------------------------------------------------------------------------
SAMPLE=class{}
function SAMPLE.new() return new({n=0,it={},ok=false,max=your.ample},SAMPLE) end

function SAMPLE.add(i,x,     pos)
  i.n = i.n + 1
  if     #i.it < i.max      then pos= #i.it + 1
  elseif rand() < #i.it/i.n then pos= #i.it * rand() end
  if pos then i.ok = false; i.it[pos//1]= x end end

function SAMPLE.all(i) if not i.ok then i.ok=true;sort(i.it)end; return i.it end
-- ------------------------------------------------------------------------------
SYM=class{}
function SYM.new(at,s)
  return new({at=at or 0,txt=s or "",has={},n=0,most=0,mode=nil},SYM) end

function SYM.__tostring(i)
  return fmt("SYM{:at %s :txt %s :mode %s :has %s}",
             i.at, i.txt, i.mode, o(i.has)) end

function SYM.add(i,x, inc)
  if x ~= "?" then
    inc = inc or 1
    i.n = i.n+inc
    i.has[x] = inc  + (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.div(i,    e)
  e=0;for _,v in pairs(i.has) do e=e - v/i.n*math.log(v/i.n,2) end; return e end

function SYM.merge(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 SYM.ranges(i,j,     t)
  t = {}
  for _,pair in pairs{{i.has,"bests"}, {j.has,"rests"}} do
    for x,inc in pairs(pair[1]) do
      t[x] = t[x] or RANGE(i,x)
      print("inc",i.txt,inc)
      t[x]:add(x, pair[2], inc) end end
  return map(t) end
```

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

fmt  = string.format
new  = setmetatable
same = function(x,...) return x end

function any(t) return t[randi(1,#t)] end

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

function copy(t,    u)
  if type(t)~="table" then return t end
  u={};for k,v in pairs(t) do u[k]=copy(v) end;return new(u,getmetatable(t)) end

function first(a,b)  return a[1] end

function firsts(a,b) return a[1] < b[1] end

function id() our.id = 1+(our.id or 0); return our.id end

function many(t,n, u) u={};for j=1,n do push(u,any(t)) end; return u end

function map(t,f,  u)
  u={};for _,v in pairs(t) do u[1+#u]=(f or same)(v) end; return u end

function main(      defaults,tasks)
  tasks = your.task=="all" and slots(go) or {your.task}
  defaults=copy(your)
  our.failures=0
  for _,x in pairs(tasks) do
    if type(our.go[x]) == "function" then our.go[x]() end
    your = copy(defaults) end
  rogues()
  return our.failures end

function merge(b4,     j,tmp,merged,one,two)
  j, tmp = 0, {}
  while j < #b4 do
    j = j + 1
    one, two = b4[j], b4[j+1]
    if two then
      merged = one.ys:merge(two.ys)
      local after=merged:div()
      local b4=xpect{one.ys,two.ys}
      if after+b4< 0.01 or after<= b4 or math.abs(after-b4)/b4 < .1 then
        j   = j+1
        one = RANGE(one.col, one.lo, two.hi, merged) end end
    push(tmp,one) end
  return #tmp==#b4 and b4 or merge(tmp) end

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

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

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 ranges(xys,col,ykind, small, dull,      one,out)
  out = {}
  xys = sort(xys, function(a,b) return a.x < b.x end)
  one = push(out, RANGE(col, xys[1].x, xys[1].x, ykind()))
```

```lua
  for j,xy in pairs(xys) do
    if   j < #xys - small   and -- enough items remaining after split
         xy.x ~= xys[j+1].x and -- next item is different (so can split here)
         one.n > small      and -- one has enough items
         one.hi - one.lo > dull -- one is not trivially small
    then one = push(out, RANGE(col, one.hi, xy.x, ykind())) end
    one:add(xy.x,  xy.y) end
  out[1].lo    = -math.huge
  out[#out].hi =  math.huge
   return out end

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

function rogues()
  for k,v in pairs(_ENV) do
    if not our.b4[k] then print("??",k,type(v)) end end end

function rows(file,      x)
  file = io.input(file)
  return function()
    x=io.read(); if x then return things(x) else io.close(file) end end end

function second(t)    return t[2] end

function seconds(a,b) return a[2] < b[2] end

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

function slots(t,u) u={};for x,_ in pairs(t) do u[1+#u]=x end;return sort(u) end

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

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

function things(x,sep,  t)
  t={};for y in x:gmatch(sep or"([^,]+)") do t[1+#t]=thing(y) end; return t end

function xpect(t,s)
 local  m,d = 0,0
 for _,z in pairs(t) do m=m+z.n; d=d+z.n*z:div() end; print(o{d=d,m=m},s or "");
 return d/m end

```

```lua
--  _            _
-- | |_ ___  ___| |_ ___
-- | __/ _ \/ __| __/ __|
-- | ||  __/\__ \ |_\__ \
--  \__\___||___/\__|___/

our.go, our.no = {},{}; go=our.go
function go.settings() print("your",o(your)) end

function go.sample() print(EGS():file(your.file)) end

function go.clone( a,b)
  a= EGS():file(your.file)
  b= a:clone(a.rows)
  asserts(#a.rows == #b.rows,"cloning rows")
  asserts(tostring(a.cols.all[1])==tostring(b.cols.all[1]),"cloning cols")
end

function go.dist(  t,a,eg1,eg2)
  a= EGS():file(your.file)
  eg1 = any(a.rows)
  print(o(eg1:col(a.cols.x)))
  t={}
  for j=1,20 do
    eg2 = any(a.rows)
    push(t, {eg1:dist(eg2,a),eg2}) end
  for _,pair in pairs(sort(t,firsts)) do
    print(o(pair[2]:col(a.cols.x)),rnd(pair[1])) end end

function go.halve(  a,b)
  a,b = EGS():file(your.file):halve()
  print(o(a:mid()))
  print(o(b:mid())) end

function go.ranges(  a,b,x,col2)
  a,b = EGS():file(your.file):halve()
  for n,col1 in pairs(a.cols.x) do
    col2 = b.cols.x[n]
    print("")
    for _, range in pairs(col1:ranges(col2)) do
      print(col1.txt, range.lo, range.hi) end end end

function go.ranges2(  a,b,x,col2)
  a,b = EGS():file(your.file):halve()
  a:ranges(b) end
--   x   = a:delta(b)
--   print(x,type(x))
--   print(">>", x.lo, x.hi)
-- end

your = settings(our.help)
os.exit( main() )
```