```lua
--------------------------------------------------------------------------------
--- vim: ts=2 sw=2 et :
local b4,help = {},[[
LESSISMORE: best or rest multi-objective optimization.
(c) 2022 Tim Menzies, timm@ieee.org
"I think the highest and lowest points are the important ones.
 Anything else is just...in between." ~ Jim Morrison

USAGE:
  alias lim="lua lessismore.lua"
  lim [OPTIONS]

OPTIONS:
  -H  --how   good or bad or novel    = good
  -m  --min    exponent of min size    = .5
  -b  --bins   max bins                = 16
  -s  --seed   random number seed      = 10019
  -S  --some   number of nums to keep  = 256
  -p  --p      exponent of distance    = 2


OPTIONS (other):
  -f  --file  where to find data       = ../etc/data/auto93.csv
  -h  --help  show help                = false
  -r  --rnd   rounding rules           = %5.2f
  -g  --go    start up action          = nothing

Usage of the works is permitted provided that this instrument is
retained with the works, so that any entity that uses the works is
notified of this instrument. DISCLAIMER:THE WORKS ARE WITHOUT WARRANTY. ]]
--------------------------------------------------------------------------------
-- ## Namespace
local the={}
local big,copy,csv,demos,discretize,dist,eg,entropy,fill_in_the,fmt,gap,is,like,lt
local map,merge,mid,mode,mu,nasa93dem,norm,num,o,oo,pdf,per,push,rand,range
local rnd,rnds,rowB4,slice,sort,some,same,sd,string2thing,sym
local NUM,SYM,RANGE,EGS,COLS,ROW
for k,_ in pairs(_ENV) do b4[k]=k end -- At end, use `b4` to find rogue vars.
--------------------------------------------------------------------------------
-- ## Coding Conventions
-- - _Separate policy from mechanism:_
--   For commonly revised parts of this processing (e.g. the name and type of data
--   columns) define a little language to support easy revision.
--   Also, all "magic parameters" that control code behavior should be part
--   of the help text. Parse that string to set those options.
--   Allow for `-h` on the command line to print that help. Allow other command
--   line flags to update those options.
-- - _Dialogue independence_:
--    Isolate and separate operating system interaction.
-- - _Test-driven development_:
--   The `go` functions store tests.
--   Tests should be silent unless they --   fail. ~tests can be
--   disabled by renaming from `go.fun` to `no.fun`. Tests should
--   return `true` if the test passes.  On exit, return number of
--   failed tests.
-- - _Write less code:_
--   "One of my most productive days was throwing away 1,000 lines of code."
--   (Ken Thompson);
--   "It is vain to do with more what can be done with less."
--   (William of Occam);
--   "Less, but better"
--   (Dieter Rams).
--   Good code is short code. If you know what is going on, the code
--   is shorter. While the code is longer, find patterns of processing
--   that combines N things into less things. Strive to write shorter.
--   Lots of short functions. Methods listed alphabetically.
--   Code 80 chars wide, or less.  Functions in 1 line,
--   if you can. Indent with two spaces. Divide code into 120 line (or
--   less) pages.  Use `i` instead of `self`.
--   Minimize use of local (exception: define all functions
--   local at top of file).
-- - _Encapsulation:_
--   Use polymorphism but no inheritance (simpler
--   debugging).  All classes get a `new` constructor.
--   Use UPPERCASE for class names.
-- - _Class,Responsibilities,Collaborators_:
--   Each class is succinctly documented as a set of collaborations
--   to fulfill some  responsibility.
-- - _Falsifiable:_
--   Code does something. It should be possible to say when that thing
--   is not happening. See external and internal metrics (Fenton).
--
-- ## About the Learning
-- - Data is stored in ROWs.
-- - Beware missing values (marked in "?") and avoid them
-- - Where possible all learning should be  incremental.
-- - Standard deviation and entropy generalized to `div` (diversity);
-- - Mean and mode generalized to `mid` (middle);
-- - Rows are created once and shared between different sets of
--   examples (so we can accumulate statistics on how we are progressing
--   inside each row).
-- - When a row is first created, it is assigned to a `base`; i.e.
--   a place to store the `lo,hi` values for all numerics.
-- - XXX tables very sueful
-- - XXX table have cols. cols are num, syms. ranges

--------------------------------------------------------------------------------
function nasa93dem()
  local vl,l,n,h,vh,xh=1,2,3,4,5,6; return {
  {"id:","center:","Year","prec","flex","resl","team","pmat","rely","data","cplx",
                              "ruse","docu","time","stor","pvol","acap","pcap","pcon",
                              "apex","plex","ltex","tool","site","sced","Kloc",
                              "Effort-","Defects-","Months-"},
   {1,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,25.9,117.6,808,15.3},
   {2,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,24.6,117.6,767,15},
   {3,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,7.7,31.2,240,10.1},
   {4,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,8.2,36,256,10.4},
   {5,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,9.7,25.2,302,11},
   {6,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,2.2,8.4,69,6.6},
   {7,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,3.5,10.8,109,7.8},
   {8,2,1982,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,66.6,352.8,2077,21},
   {9,1,1980,h,h,h,vh,h,h,l,h,n,n,xh,xh,l,h,h,n,h,n,h,h,n,n,7.5,72,226,13.6},
   {10,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,vh,n,vh,n,h,n,n,n,20,72,566,14.4},
   {11,1,1984,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,h,n,vh,n,h,n,n,n,6,24,188,9.9},
   {12,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,vh,n,vh,n,h,n,n,n,100,360,2832,25.2},
   {13,1,1985,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,n,n,vh,n,l,n,n,n,11.3,36,456,12.8},
   {14,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,h,h,h,n,h,l,vl,n,n,n,100,215,5434,30.1},
   {15,1,1983,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,h,n,vh,n,h,n,n,n,20,48,626,15.1},
   {16,1,1982,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,n,n,n,n,vl,n,n,n,100,360,4342,28},
   {17,1,1980,h,h,h,vh,n,n,l,h,n,n,n,xh,l,h,vh,n,vh,n,h,n,n,n,150,324,4868,32.5},
   {18,1,1984,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,h,n,h,n,h,n,n,n,31.5,60,986,17.6},
   {19,1,1983,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,h,n,vh,n,h,n,n,n,15,48,470,13.6},
   {20,1,1984,h,h,h,vh,n,n,l,h,n,n,n,xh,l,h,n,n,h,n,h,n,n,n,32.5,60,1276,20.8},
   {21,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,19.7,60,614,13.9},
   {22,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,66.6,300,2077,21},
   {23,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,29.5,120,920,16},
   {24,2,1986,h,h,h,vh,n,h,n,n,n,n,h,n,n,n,h,n,h,n,n,n,n,n,15,90,575,15.2},
   {25,2,1986,h,h,h,vh,n,h,n,h,n,n,n,n,n,n,h,n,h,n,n,n,n,n,38,210,1553,21.3},
   {26,2,1986,h,h,h,vh,n,n,n,n,n,n,n,n,n,n,h,n,h,n,n,n,n,n,10,48,427,12.4},
   {27,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,l,15.4,70,765,14.5},
   {28,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,l,48.5,239,2409,21.4},
   {29,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,l,16.3,82,810,14.8},
   {29,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,l,12.8,62,636,13.6},
   {31,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,l,32.6,170,1619,18.7},
   {32,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,l,35.5,192,1763,19.3},
   {33,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,5.5,18,172,9.1},
   {34,2,1987,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,10.4,50,324,11.2},
   {35,2,1987,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,14,60,437,12.4},
   {36,2,1986,h,h,h,vh,n,h,n,h,n,n,n,n,n,n,n,n,n,n,n,n,n,n,6.5,42,290,12},
   {37,2,1986,h,h,h,vh,n,n,n,h,n,n,n,n,n,n,n,n,n,n,n,n,n,n,13,60,683,14.8},
   {38,2,1986,h,h,h,vh,h,n,n,h,n,n,n,n,n,n,h,n,n,n,h,h,n,n,90,444,3343,26.7},
   {39,2,1986,h,h,h,vh,n,n,n,h,n,n,n,n,n,n,n,n,n,n,n,n,n,n,8,42,420,12.5},
   {40,2,1986,h,h,h,vh,n,n,n,h,n,n,h,n,n,n,n,n,n,n,n,n,n,n,16,114,887,16.4},
   {41,2,1980,h,h,h,vh,h,n,h,h,n,n,vh,h,l,h,h,n,n,l,h,n,n,l,177.9,1248,7998,31.5},
   {42,6,1975,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,h,n,n,n,n,n,n,n,302,2400,8543,38.4},
   {43,5,1982,h,h,h,vh,h,n,h,l,n,n,n,n,h,h,n,n,h,n,n,h,n,n,282.1,1368,9820,37.3},
   {44,5,1982,h,h,h,vh,h,h,h,l,n,n,n,n,n,h,n,n,h,n,n,n,n,n,284.7,973,8518,38.1},
   {45,5,1982,h,h,h,vh,n,h,h,n,n,n,n,n,l,n,h,n,h,n,h,n,n,n,79,400,2327,26.9},
   {46,5,1977,h,h,h,vh,l,l,n,n,n,n,n,n,l,h,vh,n,h,n,h,n,n,n,423,2400,18447,41.9},
   {47,5,1977,h,h,h,vh,h,n,n,n,n,n,n,n,l,h,vh,n,vh,l,h,n,n,n,190,420,5092,30.3},
   {48,5,1984,h,h,h,vh,h,n,n,h,n,n,n,h,n,h,n,n,h,n,h,n,n,n,47.5,252,2007,22.3},
   {49,5,1980,h,h,h,vh,l,vh,n,xh,n,n,h,h,l,n,n,n,h,n,n,h,n,n,21,107,1058,21.3},
   {50,5,1983,h,h,h,vh,l,n,h,h,n,n,vh,n,n,h,h,n,h,n,h,n,n,n,78,571.4,4815,30.5},
   {51,5,1984,h,h,h,vh,l,n,h,h,n,n,vh,n,n,h,h,n,h,n,h,n,n,n,11.4,98.8,704,15.5},
   {52,5,1985,h,h,h,vh,l,n,h,h,n,n,vh,n,n,h,h,n,h,n,h,n,n,n,19.3,155,1191,18.6},
   {53,5,1979,h,h,h,vh,l,h,n,vh,n,n,h,h,l,h,n,n,n,h,h,n,n,n,101,750,4840,32.4},
   {54,5,1979,h,h,h,vh,l,h,n,h,n,n,h,h,l,n,n,n,h,n,n,n,n,n,219,2120,11761,42.8},
   {55,5,1979,h,h,h,vh,l,h,n,h,n,n,h,h,l,n,n,n,h,n,n,n,n,n,50,370,2685,25.4},
   {56,2,1979,h,h,h,vh,h,vh,h,h,n,n,vh,vh,n,vh,vh,n,vh,n,h,h,n,l,227,1181,6293,33},
   {57,2,1977,h,h,h,vh,h,n,h,vh,n,n,n,n,l,h,vh,n,n,l,n,n,n,l,70,278,2950,20.2},
   {58,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,n,h,n,n,l,0.9,8.4,28,4.9},
   {59,6,1974,h,h,h,vh,l,vh,l,xh,n,n,xh,vh,l,h,h,n,vh,vl,h,n,n,n,980,4560,50961,96},
   {60,6,1975,h,h,h,vh,n,n,l,h,n,n,n,n,l,vh,vh,n,n,h,h,n,n,n,350,720,8547,35.7},
   {61,5,1976,h,h,h,vh,h,h,n,xh,n,n,h,h,l,h,n,n,n,h,h,h,n,n,70,458,2404,27.5},
   {62,5,1979,h,h,h,vh,h,h,n,xh,n,n,h,h,l,h,n,n,n,h,h,h,n,n,271,2460,9308,43.4},
   {63,5,1971,h,h,h,vh,n,n,n,n,n,n,n,n,l,h,h,n,h,n,h,n,n,n,90,162,2743,25},
   {64,5,1980,h,h,h,vh,n,n,n,n,n,n,n,n,l,h,h,n,h,n,h,n,n,n,40,150,1219,18.9},
   {65,5,1979,h,h,h,vh,n,h,n,h,n,n,h,n,l,h,h,n,h,n,h,n,n,n,137,636,4210,32.2},
   {66,5,1977,h,h,h,vh,n,h,n,h,n,n,h,n,h,h,h,n,h,n,h,n,n,n,150,882,5848,36.2},
   {67,5,1976,h,h,h,vh,n,vh,n,h,n,n,h,n,l,h,h,n,h,n,h,n,n,n,339,444,8477,45.9},
   {68,5,1983,h,h,h,vh,n,l,h,l,n,n,n,n,h,h,h,n,h,n,h,n,n,n,240,192,10313,37.1},
   {69,5,1978,h,h,h,vh,l,h,n,h,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,144,576,6129,28.8},
   {70,5,1979,h,h,h,vh,l,n,l,n,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,151,432,6136,26.2},
   {71,5,1979,h,h,h,vh,l,n,l,h,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,34,72,1555,16.2},
   {72,5,1979,h,h,h,vh,l,n,n,h,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,98,300,4907,24.4},
   {73,5,1979,h,h,h,vh,l,n,n,h,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,85,300,4256,23.2},
   {74,5,1982,h,h,h,vh,l,n,l,n,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,20,240,813,12.8},
   {75,5,1978,h,h,h,vh,l,n,l,n,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,111,600,4511,23.5},
   {76,5,1978,h,h,h,vh,l,h,vh,h,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,162,756,7553,32.4},
   {77,5,1978,h,h,h,vh,l,h,h,vh,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,352,1200,17597,42.9},
   {78,5,1979,h,h,h,vh,l,h,n,vh,n,n,n,vh,l,h,h,n,h,h,h,n,n,l,165,97,7867,31.5},
   {79,5,1984,h,h,h,vh,h,h,n,vh,n,n,h,h,l,h,n,n,n,h,h,n,n,n,60,409,2004,24.9},
   {80,5,1984,h,h,h,vh,h,h,n,vh,n,n,h,h,l,h,n,n,n,h,h,n,n,n,100,703,3340,29.6},
   {81,2,1980,h,h,h,vh,n,h,vh,vh,n,n,xh,xh,h,n,n,n,n,l,l,n,n,n,32,1350,2984,33.6},
   {82,2,1980,h,h,h,vh,h,h,h,h,n,n,vh,xh,h,h,h,n,h,h,h,n,n,n,53,480,2227,28.8},
   {83,3,1977,h,h,h,vh,h,h,l,vh,n,n,vh,xh,l,vh,vh,n,vh,vl,vl,h,n,n,41,599,1594,23},
   {84,3,1977,h,h,h,vh,h,h,l,vh,n,n,vh,xh,l,vh,vh,n,vh,vl,vl,h,n,n,24,430,933,19.2},
   {85,5,1977,h,h,h,vh,h,vh,h,vh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,n,165,4178.2,6266,47},
   {86,5,1977,h,h,h,vh,h,vh,h,vh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,n,65,1772.5,2468,34.5},
   {87,5,1977,h,h,h,vh,h,vh,h,vh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,n,70,1645.9,2658,35.4},
   {88,5,1977,h,h,h,vh,h,vh,h,xh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,n,50,1924.5,2102,34.2},
   {89,5,1982,h,h,h,vh,l,vh,l,vh,n,n,vh,xh,l,h,n,n,l,vl,l,h,n,n,7.25,648,406,15.6},
   {90,5,1980,h,h,h,vh,h,vh,h,vh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,n,233,8211,8848,53.1},
   {91,2,1983,h,h,h,vh,n,h,n,vh,n,n,vh,vh,h,n,n,n,n,l,l,n,n,n,16.3,480,1253,21.5},
   {92,2,1983,h,h,h,vh,n,h,n,vh,n,n,vh,vh,h,n,n,n,n,l,l,n,n,n,6.2,12,477,15.4},
   {93,2,1983,h,h,h,vh,n,h,n,vh,n,n,vh,vh,h,n,n,n,n,l,l,n,n,n,3,38,231,12}} end
--------------------------------------------------------------------------------
-- ## Utils
-- Misc
big=math.huge
rand=math.random
fmt=string.format
same = function(x) return x end

-- Sorting
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

-- Query and update
function map(t,f, u)  u={};for k,v in pairs(t) do u[1+#u]=f(v) end; return u end
function push(t,x)    t[1+#t]=x; return x end
function slice(t,i,j,k,     u)
  i,j = (i or 1)//1, (j or #t)//1
  k   = (k and (j-i)/k or 1)//1
  u={}; for n=i,j,k do u[1+#u] = t[n] end return u end

-- "Strings 2 things" coercion.
function string2thing(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(csvfile)
  csvfile = io.input(csvfile)
  return function(line, row)
    line=io.read()
    if not line then io.close(csvfile) else
      row={}; for x in line:gmatch("([^,]+)") do push(row,string2thing(x)) end
      return row end end end

-- "Things 2 strings" coercion.
function oo(t) print(o(t)) end
function o(t,    u)
  if #t>0 then return "{"..table.concat(map(t,tostring)," ").."}" else
    u={}; for k,v in pairs(t) do u[1+#u] = fmt(":%s %s",k,v) end
    return (t.is or "").."{"..table.concat(sort(u)," ").."}" end end

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

-- Convert help string to a table. Check command line for any updates.
function fill_in_the(shortFlag,longFlag,slot,x)
  for n,flag in ipairs(arg) do
    if flag==shortFlag or flag==longFlag then
      x = x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
  the[slot] = string2thing(x) end

-- Run demos, each time resetting settings and random seed. Return #failures.
local go,no={},{} -- place to store enabled and disabled tests
function demos(    fails,names,defaults,status)
  fails=0     -- this code will return number of failures
  names, defaults = {},{}
  for k,f in pairs(go) do if type(f)=="function" then push(names,k) end end
  for k,v in pairs(the) do defaults[k]=v end
  if go[the.go] then names={the.go} end
  for _,one in pairs(sort(names))  do         -- for all we want to do
    for k,v in pairs(defaults) do the[k]=v end -- set settings to defaults
    math.randomseed(the.seed or 10019)         -- reset random number seed
    io.stderr:write(".")
    status = go[one]()                         -- run demo
    if status ~= true then
      print("-- Error",one,status)
      fails = fails + 1 end end                -- update fails
  return fails end                             -- return total failure count

-- Polymorphic objects.
function is(name,    t,new)
  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
```

```
------------------------------------------------------------------------------
-- ## Objects
COLS,EGS,NUM,RANGE,ROW,SYM=is"COLS",is"EGS",is"NUM",is"RANGE",is"SYM",is"ROW"
------------------------------------------------------------------------------
-- ## NUM
-- - For a stream of `add`itions, incrementally maintain `mu,sd`.
-- - `Norm`alize data for distance and discretization calcs
--    (see `dist` and `range`).
-- - Comment on `like`lihood that something belongs to this distribution.
function NUM.new(i,at,txt)
  i.at=at or 0; i.txt=txt or ""; i.lo,i.hi=big, -big
  i.n,i.mu,i.m2,i.sd = 0,0,0,0,0;  i.w=(txt or""):find"-$" and -1 or 1 end

function NUM.add(i,x,   d)
  if x=="?" then return x end
  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.sd = (i.m2<0 or i.n<2) and 0 or ((i.m2/(i.n - 1))^0.5)
  i.lo = math.min(i.lo,x)
  i.hi = math.max(i.hi,x) end

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

function NUM.like(i,x,_,       e)
  return (x < i.mu - 4*i.sd and 0 or x > i.mu + 4*i.sd and 0 or
    2.7183^(-(x - i.mu)^2 / (z + 2*i.sd^2))/(z + (math.pi*2*i.sd^2)^.5)) end

function NUM.merge(i,ranges,min,      a,b,c,j,n,tmp)
  function expand(t)
    if #t<2 then return {} end
    for j=2,#t do t[j].lo=t[j-1].hi end
    t[1].x.lo, t[#t].x.hi= -big,big
    return t
  end --------------
  j,n,tmp = 1,#ranges,{}
  while j<=n do
    a, b = ranges[j], ranges[j+1]
    if b then c = a:merge(b,min); if c then a,j = c,j+1 end end
    tmp[#tmp+1] = a
    j = j+1 end
  return #tmp==#ranges and expand(tmp) or i:merge(tmp,min) 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+1/big) end

function NUM.range(i,x,n,  b) b=(i.hi-i.lo)/n; return math.floor(x/b+0.5)*b end
------------------------------------------------------------------------------
-- ## SYM
-- - For a stream of `add`itions, incrementally maintain count of `all` symbols.
-- - Using that info, report `dist`, mode (`mid`) symbol, and entropy
--   (`div`) of this distribution.
-- - Comment on `like`lihood that something belongs to this distribution.
-- - Discretization of a symbol just returns that sym (`range`).
function SYM.new(i,at,txt) i.at=at or 0; i.txt=txt or ""; i.n,i.all = 0,{} end

function SYM.add(i,x,n)
  if x=="?" then return x end
  n = n or 1
  i.n=i.n+n; i.all[x] = n + (i.all[x] or 0) end

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

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

function SYM.like(i,x,prior) return ((c.all[x] or 0)+the.m*prior)/(c.n+the.m) end

function SYM.merge(i,ranges,min) return ranges end

function SYM.mid(i)
  m=0; for y,n in pairs(i.all) do if n>m then m,x=n,y end end; return x end

function SYM.range(i,x,_) return x end
```

```
------------------------------------------------------------------------------
-- ## RANGE
-- - For a stream of `add`itions, incrementally maintain counts of `x` and `y`.
-- - Summarize `x` as the `lo,hi` seen so far and summarize `y` in `SYM` counts
--   in `y.all` (and get counts there using `of`).
-- - Support range sorting (`__lt`) and printing (`__tostring`).
-- - Check if this range's `x` values `select`s for a particular row.
-- - `Merge` adjacent ranges if the entropy of the whole is less than the parts.
function RANGE.new(i,col,lo,hi,y)
  i.col, i.x, i.y = col, {lo=lo or big, hi=hi or -big}, (y or  SYM()) end

function RANGE.__lt(i,j) return i.x.lo < j.x.lo end

function RANGE.__tostring(i)
  local x, lo, hi = i.col.txt, i.x.lo, i.x.hi
  if     lo ==  hi  then return fmt("%s == %s",x, lo)
  elseif hi ==  big then return fmt("%s >= %s",x, lo)
  elseif lo == -big then return fmt("%s < %s", x, hi)
  else                   return fmt("%s <= %s < %s",lo,x,hi) end end

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

function RANGE.merge(i,j,n0,    k)
  k = SYM(i.col.at, i.col.txt)
  for x,n in pairs(i.y.all) do k:add(x,n) end
  for x,n in pairs(j.y.all) do k:add(x,n) end
  if i.y.n<(n0 or 0) or j.y.n<(n0 or 0) or (
     (i.y:div(i)*i.y.n + j.y:div()*j.y.n)/k.n >= .99*k:div())
  then return RANGE(i.col, i.x.lo, j.x.hi, k) end end

function RANGE.of(i,x) return i.y.all[x] or 0 end

function RANGE.score(i,goal,B,R, how)
  how={}
  how.good= function(b,r) return ((b<r or b+r < .05) and 0) or b^2/(b+r) end
  how.bad=  function(b,r) return ((r<b or b+r < .05) and 0) or r^2/(b+r)  end
  how.novel=function(b,r) return 1/(b+r) end
  b, r, z = 0, 0, 1/big
  for x,n in pairs(i.y.all) do
    if x==goal then b = b+n else r=r+n end end
  return how[the.how or "good"](b/(B+z), r/(R+z)) end

function RANGE.selects(i,t,     x)
  t = t.cells and t.cells or t
  x = t[i.at]
  return x=="?" or (i.x.lo==i.x.hi and i.x.lo==x) or (i.x.lo<=x and x<i.x.hi)end
------------------------------------------------------------------------------
-- ## ROW
-- - Using knowledge `of` the geometry of the data, support distance calcs
-- i  (`__sub` and `around`) as well as multi-objective ranking (`__lt`).
function ROW.new(i,eg, cells) i.of,i.cells = eg,cells end

function ROW.__lt(i,j,     s1,s2,e,y,a,b)
  y = i.of.cols.y
  s1, s2, e = 0, 0,  math.exp(1)
  for _,col in pairs(y) do
     a  = col:norm(i.cells[col.at])
     b  = col:norm(j.cells[col.at])
     s1 = s1 - e^(col.w * (a - b) / #y)
     s2 = s2 - e^(col.w * (b - a) / #y) end
  return s1/#y < s2/#y end

function ROW.__sub(i,j)
  for _,col in pairs(i.of.cols.x) do
    a,b = i.cells[col.at], j.cells[col.at]
    inc = a=="?" and b=="?" and 1 or col:dist(a,b)
    d   = d + inc^the.p end
  return (d / (#i.of.cols.x)) ^ (1/the.p) end

function ROW.around(i,rows)
  return sort(map(rows or i.of.rows, function(j) return {dist=i-j,row=j} end),
              lt"dist") end
```

```
------------------------------------------------------------------------------
-- ## COLS
-- - Factory for converting column `names` to `NUM`s ad `SYM`s.
-- - Store all columns in -- `all`, and for all columns we are not skipping,
--   store the independent and dependent columns distributions in `x` and `y`.
function COLS.new(i,names,     head,row,col)
  i.names=names; i.all={}; i.y={}; i.x={}
  for at,txt in pairs(names) do
    col       = push(i.all, (txt:find"^[A-Z]" and NUM or SYM)(at, txt))
    col.goalp = txt:find"[!+-]$" and true or false
    if not txt:find":$" then
      if txt:find"!$" then i.klass=col end
      push(col.goalp and i.y or i.x, col) end end end
-----------------------------------------------------------------------------
-- ## EGS
-- - For a stream of `add`itions, incrementally store rows, summarized in `cols`.
-- - When `add`ing, build new rows for new data. Otherwise reuse rows across
--   multiple sets of examples.
-- - Supporting `copy`ing of this structure, without or without rows of data.
-- - Report how much this set of examples `like` a new row.
-- - Discretize columns as `ranges` that distinguish two sets of rows
--   (merging irrelevant distinctions).
-- - Summarize the `mid`point of these examples.
function EGS.new(i,names) i.rows,i.cols = {}, COLS(names) end

function EGS.add(i,row,    cells)
  cells = push(i.rows, row.cells and row or ROW(i,row)).cells
  for n,col in pairs(i.cols.all) do col:add(cells[n]) end end

function EGS.copy(i,rows,  j)
  j=EGS(i.cols.names); for _,r in pairs(rows or {}) do j:add(r) end;return j end

function EGS.like(i,t,overall, nHypotheses,      c)
  prior = (#i.rows + the.k) / (overall + the.k * nHypotheses)
  like  = math.log(prior)
  for at,x in pairs(t) do
    c=i.cols.all.at[at]
    if x~="?" and not c.goalp then
      like = math.log(col:like(x)) + like end end
  return like end

function EGS.load(src,   i)
  if    src==nil or type(src)=="string"
  then for   row in csv(src)   do if i then i:add(row) else i=EGS(row)end end
  else for _,row in pairs(src) do if i then i:add(row) else i=EGS(row)end end end
  return i end

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

function EGS.ranges(i,yes,no,   out,x,bin,tmp,score)
  out={}
  for _,col in pairs(i.cols.x) do  -- for each x col
    tmp = {}                       -- find ranges that distinguish yes and no
    for _,what in pairs{{rows=yes, klass=true}, {rows=no, klass=false}} do
      for _,row in pairs(what.rows) do x = row.cells[col.at]
        if x~="?" then
          bin      = col:range(x,the.bins)
          tmp[bin] = tmp[bin] or RANGE(col,x,x)
          tmp[bin]:add(x, what.klass) end end end
    tmp = map(tmp,same) --  a hack. makes tmp sortable (has consecutive indexes)
    for _,range in pairs(col:merge(sort(tmp),(#yes+#no)^the.min)) do
      push(out,range) end end
  score = function(range) return range:score(true,#yes, #no) end
  return sort(out,score) end
```

```lua
------------------------------------------------------------------------------
-- ## Code for tests and demos

-- Simple stuff
function go.the()     return type(the.bins)=="number" end
function go.sort(  t) return 0==sort({100,3,4,2,10,0})[1] end
function go.slice( t,u)
  t = {10,20,30,40,50,60,70,80,90,100,110,120,130,140}
  u = slice(t,3,#t,3)
  t = slice(t,3,5)
  return #t==3 and #u==4 end

function go.num(     n,mu,sd)
  n, mu, sd = NUM(), 10, 1
  for i=1,10^4 do
    n:add(mu+sd*math.sqrt(-2*math.log(rand()))*math.cos(2*math.pi*rand())) end
  return math.abs(n.mu - mu) < 0.05 and math.abs(n.sd - sd) < 0.5 end

-- Can we read rows off the disk?
function go.rows( n,m)
  m,n=0,0; for row in csv(the.file) do m=m+1; n=n+#row; end; return n/m==8 end

-- Can we turn a list of names into columns?
function go.cols(  i)
  i=COLS{"name","Age","ShoeSize-"}
  return i.y[1].w == -1 end

-- Can we read data, summazized as columns?
function go.egs(  it)
  it=EGS.load(nasa93dem())
  return math.abs(it.cols.y[1].mu - 624) < 1 end
  --for _,row in pairs(nasa93dem())do oo(row) end end
  --it = EGS.load(the.file); return math.abs(2970 - it.cols.y[1].mu) < 1 end

-- Does discretization work?
function go.ranges(  it,n,best,rest,min)
  --it = EGS.load(the.file)
  print(the.how)
  it=EGS.load(nasa93dem())
  print("all",o(rnds(it:mid())))
  it.rows = sort(it.rows)
  for j,row in pairs(sort(it.rows)) do row.klass = 1+j//(#it.rows*.35/6) end
  n = (#it.rows)^.5
  best,rest = slice(it.rows,1,n), slice(it.rows, n+1, #it.rows, 3*n)
  print("best",#best,o(rnds(it:copy(best):mid())))
  print("rest",#rest,o(rnds(it:copy(rest):mid())))
  tmp={}; for _,ranges in pairs(it:ranges(best,rest)) do
    for at,range in pairs(ranges) do
    push(tmp,range).val= range:score(true,#best,#rest) end end
  for _,range in pairs(sort(tmp,lt"val")) do print(range.val, range) end
  --oo(a:mid())
  --oo(b:mid())
  return math.abs(2970 - it.cols.y[1].mu) < 1 end
```

```lua
--------------------------------------------------------------------------------
-- ## Main

-- - Parse help text for flags and defaults, check CLI for updates.
-- - Maybe print the help (with some pretty colors).
-- - Run the demos.
-- - Check for rogue vars.
-- - Exit, reporting number of failures.
help:gsub("\n ([-][^%s]+)[%s]+([-][-]([^%s]+))[^\n]*%s([^%s]+)",fill_in_the)
if the.help then
  print(help:gsub("%u%u+", "\27[31m%1\27[0m")
            :gsub("(%s)([-][-]?[^%s]+)(%s)","%1\27[33m%2\27[0m%3"),"")
else
  local fails = demos()
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end  end
  os.exit(fails) end
```