```lua
-- <h3>L5 = A Little Light Learner Lab, in LUA</h3>
-- <img src=l5.png align=left width=300>
--
-- [&copy; 2022](#copyright) Tim Menzies<br>[Contribute](#contribute)<br>
--
-- Here, we write the _most_ learners in the _least_ code.
-- Each learner is a few lines of code (since they share an
-- underlying code base).
--
-- Why LUA? Three reasons.
--
-- __ONE__:<br>LUA supports simple teaching
-- (less than 2 dozen keywords). Heck, children use it to code up their own games.
--
-- __TWO__:<br>The great secret is that LUA==LISP (ish). LUA supports many advance
d programming
-- techniques (first class
-- objects, functional programming, etc) without  (**L**ots of (**I**nfuriating (*
*S**illy
-- (**P**arenthesis)))).  For example, the entire object system used here is just
five lines of code
-- (see **is()**).
--
-- __THREE__:<br>my standard assignment is "here is  a worked solution,
-- now code it up in any other language". So with LUA/L5 I can give students an
-- succinct executable specification that demonstrates numerous recommended coding
-- practices (for learning and for scripting).
-- And then they can still code in their language du jour.
--
-- e.g. __Pass1:__ Recursively bi-cluster, sample 1 point per cluster,
-- prune cluster with worst point. __Pass2:__ Do it again, using the better
-- things found in Pass1. __Pass3:__ Report rules that selects for the
-- "good" found in Pass2.
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
local add,big,col,csv,fmt,fyi,id,is,klass,lt,map,oo
local per,push, rand, ranges,read, result, rnd, seed, splice, str
local help=[[
L5: a little light learner lab in LUA
(c) 2022 Tim Menzies, timm@ieee.org, BSD2 license

INSTALL: requires: lua 5.4+
         download: l5.lua  and data/* from github.com/timm/l5
         test    : lua l5.lua -f data/auto93.csv; echo $? # expect "0"

USAGE: lua l5.lua [OPTIONS]
                                              defaults
                                              ~~~~~~~~
  -S  --Seed  random number seed              = 10019
  -H  --How   optimize for (helps,hurts,tabu) = helps
  -b  --bins  number of bins                  = 16
  -m  --min   min1 size (for pass1)           = .5
  -M  --Min   min2 size (for pass2)           = 10
  -p  --p     distance coefficient            = 2
  -s  --some  sample size                     = 512

OPTIONS (other):
  -f  --file  csv file with data = data/auto93.csv
  -g  --go    start up action    = nothing
  -v  --verbose show details     = false
  -h  --help  show help          = false]]
-- ## Convert help text to settings

-- __read(str:str) :bool | int | str__ <br> String to thing.
function read(str)
  str = str:match"^%s*(.-)%s*$"
  if str=="true" then return true elseif str=="false" then return false end
  return math.tointeger(str) or tonumber(str) or str  end

-- (1) parse `help`.<br>(2) make `THE` settings.<br>(3) Also make a `backup`.
local THE, backup = {}, {}
help:gsub(" [-][-]([^%s]+)[^\n]*%s([^%s]+)",function(key,x)
  for n,flag in ipairs(arg) do
    if flag==("-"..key:sub(1,1)) or flag==("--"..key) then
       x= x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
  x = read(x)
  backup[key] = x
  THE[key] = x end)

-- If `-h` was used on command line, pretty print help text (then exit).
if THE.help then os.exit(print(help:gsub("[%u][%u%d]+","\27[1;31m%1\27[0m"))) end

-- ##  Define Classes

-- __str(i:any) :str__
-- Make pretty print string from tables. Print  slots of associative arrays in sor
ted order.
-- To actually print this string, use `oo(i)` (see below).
function str(i)
  local j
  if type(i)~="table" then return tostring(i) end
  if #i> 0            then return table.concat(map(i,tostring),", ") end
  j={}; for k,v in pairs(i) do j[1+#j] = string.format(":%s %s",k,v) end
  table.sort(j)
  return (i.is or "").."{"..table.concat(j," ").."}" end

-- __is(name:str) :klass__
-- Object creation.<br>(1) Link to pretty print.<br>(2) Assign a unique id.
-- (3) Link new object to the class.<br>Map klass(i,...) to klass.new(...).
local _id=0
function is(name,    t)
  local function new(kl,...)
    _id = _id+1
    local x=setmetatable({id=_id},kl); kl.new(x,...); return x end
  t = {__tostring=str, is=name}; t.__index=t
  return setmetatable(t, {__call=new}) end

-- Make our classes.<br>(1) Data is stored as set of ROW.
-- (2) ROWS are containers for ROW. <br>(3) Columns are summarized
-- as SYMbolics or NUMerics.<br>(4) SOME is a helper class for NUM.
-- (5) RANGE is a helper class for EGS.
local ROW,ROWS,SYM   = is"ROW",is"ROWS",is"SYM"
local NUM,RANGE,SOME = is"NUM",is"RANGE",is"SOME"

-- ##  SOME methods
-- If we keep more than
-- `THE.some` items then SOME replaces old items with the new old items.

-- __col(i:column, has:t, ?at:int=1, ?txt:str="")__
-- For SOME (and NUM and SYM), new columns have a container `has` and appear in
-- column `at` and have name `txt`. If a column name ends in `-`, set its weight
-- to -1.
function col(i,has,at,txt)
  i.n, i.at, i.txt = 0, at or 0, txt or ""
  i.w= i.txt:find"-$" and -1 or 1
  i.has = has end

-- __add(i:column, x:any, nil | inc:int=1, fun:function):x)__
-- Don't add missing values. When you add something, inc the `i.n` count.
function add(i,x,inc,fun)
  if x ~= "?" then
    inc = inc or 1
    i.n = i.n + inc
    fun() end
  return  end

-- __SOME(?at:int=1, ?txt:str="") :SOME__
function SOME.new(i, ...) col(i,{},...); i.ok=false; end
-- __SOME:add(x:num):x__
function SOME.add(i,x)
  return add(i,x,1,function(     a)
    a = i.has
    if     #a     < THE.some     then i.ok=false; push(a,x)
  elseif rand() < THE.some/i.n then i.ok=false; a[rand(#a)]=x end end) end

-- __SOME:sorted(): [num]*__
-- Return the contents, sorted.
function SOME.sorted(i,  a)
if not i.ok then table.sort(i.has) end; i.ok=true; return i.has end

-- ##  NUM methods

-- (1) Incrementally update a  sample of numbers including its mean `mu`,
--     min `lo` and max `hi`.
-- (2) Knows how to calculate the __div__ ersity of a sample (a.k.a.
--     standard deviation).

-- __NUM(?at:int=1, ?txt:str="") :NUM__
function NUM.new(i, ...) col(i,SOME(),...); i.mu,i.lo,i.hi=0,big,-big end
-- __NUM:add(x:num):x__
function NUM.add(i,x)
return add(i,x,1,function(     d)
  i.has:add(x)
  d = x - i.mu
  i.mu = i.mu + d/i.n
  i.hi = math.max(x, i.hi); i.lo=math.min(x, i.lo) end ) end

-- __NUM:clone():NUM__  <br> Duplicate structure
function NUM.clone(i)    return NUM(i.at, i.txt) end

-- __NUM:merge(j:num):NUM__ <br> Combine two NUMs.
function NUM.merge(i,j,      k)
  local k = NUM(i.at, i.txt)
  for _,x in pairs(i.has.has) do k:add(x) end
  for _,x in pairs(j.has.has) do k:add(x) end
  return k end

-- __NUM:mid():num__ <br>mid is `mu`.
function NUM.mid(i,p) return rnd(i.mu,p or 3) end
-- __NUM:div():num__ <br>div is entropy
function NUM.div(i, a)
  a=i.has:sorted(); return (per(a, .9) - per(a, .1))/2.56 end

-- __NUM:bin(x:num):num__<br>NUMs get discretized to bins of size `(hi - lo)/THE.b
ins`.
function NUM.bin(i,x,   b)
  b = (i.hi - i.lo)/THE.bins; return math.floor(x/b+.5)*b end

-- __NUM:norm(x:num):num__<br>Normalize `x` 0..1 for `lo`..`hi`.
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

-- ## SYM methods

-- Incrementally update a  sample of numbers including its mode
-- and **div**ersity (a.k.a. entropy)
function SYM.new( i, ...) col(i,{},...);     i.most, i.mode=0,nil end

-- __SYM.clone():SYM__<br>Duplicate the structure.
function SYM.clone(i) return SYM(i.at, i.txt) end

-- __NUM:add(x:any):x__
function SYM.add(i,x,inc)
return add(i,x,inc, function()
  i.has[x] = (inc or 1) + (i.has[x] or 0)
  if i.has[x] > i.most then i.most,i.mode = i.has[x],x end end) end

-- __SYM:merge(j:num):SYM__ <br> Combine two NUMs.
function SYM.merge(i,j,      k)
  local k = SYM(i.at, i.txt)
  for x,n in pairs(i.has) do k:add(x,n) end
  for x,n in pairs(j.has) do k:add(x,n) end
  return k end

-- __SYM:mid():any__ <br>Mode.
function SYM.mid(i,...) return i.mode end
-- __SYM:div():float__ <br>Entropy.
function SYM.div(i,      e)
  e=0;for k,n in pairs(i.has) do if n>0 then e=e-n/i.n*math.log(n/i.n,2)end end
  return e end

-- __SYM:bin(x:any):x__<br>SYMs get discretized to themselves.
function SYM.bin(i,x) return x end

-- __SYM:score(want:any, wants:int, donts:init):float__ <br>SYMs get discretized t
o themselves.
function SYM.score(i,want, wants,donts)
  local b, r, z, how = 0, 0, 1/big, {}
  how.helps= function(b,r) return (b<r or b+r < .05) and 0 or b^2/(b+r) end
  how.hurts= function(b,r) return (r<b or b+r < .05) and 0 or r^2/(b+r) end
  how.tabu = function(b,r) return 1/(b+r+z) end
  for v,n in pairs(i.has) do if v==want then b = b+n else r=r+n end end
  return how[THE.How](b/(wants+z), r/(donts+z)) end

-- ##  ROW methods

-- The `cells` of one ROW store one record of data (one ROW per record). If ever w
e read the y-values then that
-- ROW is `evaluated`. For many tasks, data needs to be __normalized__ in which ca
se
-- we need to know the space `of` data that holds this data.
function ROW.new(i,of,cells) i.of,i.cells,i.evaluated = of,cells,false end

-- <b>i:ROW < j:ROW</b> <br>`i` comes before `j` if its y-values are better.
-- This is Zitzler's continuous domination predicate. In summary, it is a small
-- "what-if" study that walks from one way, then the other way, from one
-- example to another. The best row is the one that looses the least.
function ROW.__lt(i,j,        n,s1,s2,v1,v2)
  i.evaluated = true
  j.evaluated = true
  s1, s2, n = 0, 0, #i.of.ys
  for _,col in pairs(i.of.ys) do
    v1,v2 = col:norm(i.cells[col.at]), col:norm(j.cells[col.at])
    s1    = s1 - 2.7183^(col.w * (v1 - v2) / n)
    s2    = s2 - 2.7183^(col.w * (v2 - v1) / n) end
  return s1/n < s2/n end

-- __ROW:within(range):bool__
function ROW.within(i,range,          lo,hi,at,v)
   lo, hi, at = range.xlo, range.xhi, range.ys.at
   v = i.cells[at]
   return  v=="?" or lo==hi and v==lo or lo<=v and v<hi end
```

```lua

-- ## ROWS methods
-- Sets of ROWs are stored in ROWS. ROWS summarize columns and those summarizes
-- are stored in `cols`. For convenience, all the columns we are not skipping
-- are also contained into the goals and non-goals `xs`, `ys`.

-- __ROWS(src:str | tab):ROWS__<br>Load in examples from a file string, or a list or rows.
function ROWS.new(i,src)
  i.has={}; i.cols={}; i.xs={}; i.ys={}; i.names={}
  if type(src)=="string" then for   row in csv(  src) do i:add(row) end
                         else for _,row in pairs(src) do i:add(row) end end end

-- __ROWS:clone(?with:tab):ROWS__
-- Duplicate structure, then maybe fill it in  `with` some data.
function ROWS.clone(i,with,    j)
  j=ROWS({i.names}); for _,r in pairs(with or {}) do j:add(r) end; return j end

-- __ROWS:add(row: (tab| ROW))__
-- If this is the first row, create the column summaries.
-- Else, if this is not a ROW, then make  one and set its `of` to `i`.
-- Else, add this row to `ROWS.has`.
-- When adding a row, update the column summaries.
function ROWS.add(i,row)
  local function header(   col)
    i.names = row
    for at,s in pairs(row) do
      col = push(i.cols, (s:find"^[A-Z]" and NUM or SYM)(at,s))
      if not s:find":$" then
        if s:find"!$" then i.klass = col end
        push(s:find"[!+-]$" and i.ys or i.xs, col) end end
  end -------------------------------
  if #i.cols==0 then header(row) else
    row = push(i.has, row.cells and row or ROW(i,row))
    for _,col in pairs(i.cols) do col:add(row.cells[col.at]) end end end

-- __ROWS:bestRest()__<br>Return the rows, divided into the best or rest.
function ROWS.bestRest(i,  n,m)
  table.sort(i.has)
  n = #i.has
  m = n^THE.min
  return splice(i.has, 1,  m), splice(i.has, n - m) end

-- __ROWS:mid(?p:int=3) :tab__<br>Return the `mid` of the goal columns.
-- Round numerics to `p` places.
function ROWS.mid(i,p,    t)
  t={}; for _,col in pairs(i.ys) do t[col.txt]=col:mid(p) end; return t end

-- __ROWS:splits(best0:[ROW], rests:[ROW]):[ROW],[ROW],RANGE}__
-- Supervised discretization: return ranges that are most difference in `bests0` and `rests0`.
function ROWS.splits(i,bests0,rests0)
 print(#bests0, #rests0)
  most,range,range1,score = -1
  for _,col in pairs(i.xs) do
    print(col)
    for _,range0 in ranges(col,bests0,rests0) do
      score = range0:score(1,#bests0,#rests0)
      if score>most then most,range1 = score,range0 end end end
  local bests1, rests1 = {},{}
  for _,rows in pairs{bests0,rests0} do
    for _,row in pairs(rows) do
      push(row:within(range1) and bests1 or rests1, row) end end
  return bests1, rests1, range1 end

-- __ROWS:contrast(best0:[row], rests0:[row]):[row]__
-- Recursively find ranges that selects for the best rows.
function ROWS.contrast(i,bests0,rests0,    hows,stop)
  stop = stop or #bests0/4
  hows = hows or {}
  print(1)
  bests1, rests1,range = i:splits(bests0, rests0)
  if (#bests0 + #rests0) > stop and (#bests1 < #bests0 or #rests1 < #rests0) then
    push(hows,range)
    return i:contrast(bests1, rests1, hows, stop) end
  return hows0,bests0 end

-- ## RANGE methods

-- Given some x values running from `xlo` to `xhi`, store the
-- `ys`  y values seen
function RANGE.new(i, xlo, xhi, ys) i.xlo, i.xhi, i.ys = xlo, xhi, ys end

-- __RANGE:add(x:atom, y:atom)__
function RANGE.add(i,x,y)
  if x < i.xlo then i.xlo = x end -- works for string or num
  if x > i.xhi then i.xhi = x end -- works for string or num
  i.ys:add(y) end

-- **RANGE:__tostring()**<br>Pretty print.
function RANGE.__tostring(i)
  local x, lo, hi = i.ys.txt, i.xlo, i.xhi
  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


-- **ranges(col: NUM | SYM, rows1:[row], rows2:[row], ...):[RANGE]**
-- This function generates ranges.
-- Return a useful way to divide the values seen in this column,
-- in these different rows.
function ranges(col, ...)
  -- For numerics, **xpand** the ranges to cover the whole number line.
  local function xpand(t)  --  extend ranges to cover whole number line
    for j=2,#t do t[j].xlo = t[j-1].xhi end
    t[1].xlo, t[#t].xhi = -big, big
    return t end
  -- **Merged** returns "nil" if the merge would actually complicate things
  local function merged(i,j,min,      k)
    k = i:merge(j)
    if i.n < min or j.n < min or k:div()<=(i.n*i:div() + j.n*j:div())/k.n then
      return k end end
   -- **Merge** adjacent ranges if     they have too few examples, or
   -- the whole is simpler than that parts. Keep merging, until we
   -- can't find anything else to merge.
  local function merge(b4,min,      t,j,a,b,c)
    t,j = {},1
    while j <= #b4 do
      a, b = b4[j], b4[j+1]
      if b then
         c = merged(a.ys, b.ys, min) -- merge small bins that are to complex
         if c then
           j = j + 1
           a = RANGE(a.xlo, b.xhi, c) end end
      t[#t+1] = a
      j = j + 1 end
    return #b4 == #t and t or merge(t,min)  -- (3)
  end ----------------------------
  --  For discretized values at `col.at`, create ranges that count how
  -- often those values  appear in a set of rows (sorted 1,... for best...worst).

  local known,out,n,v,x = {},{}, 0
  for klass,rows in pairs{...} do -- for each set..
    n = n + #rows
    for _,row in pairs(rows) do    -- for each row...
      v = row.cells[col.at]
      if v ~= "?" then               -- count how often we see some value
        x = col:bin(v)                 -- accumulated into a few bins
        -- The next line idiom means "known[x]" exists, and is stored in "out".
        known[x] = known[x] or push(out,RANGE(v, v, SYM(col.at,col.txt)))
        known[x]:add(v,klass) end end end   -- do the counting
  table.sort(out,lt("xlo"))
  out= col.is=="NUM" and xpand(merge(out, n^THE.min)) or out
  return #out < 2 and {} or out -- less than 2 ranges? then no splits found!
end -- ranges

-- ## Functions

-- Large number
big = math.huge
-- __csv(csvfile:str)__ :<br>Iterator. Return one table per line, split on ",".
function csv(csvfile)
  csvfile = io.input(csvfile)
  return function(s, t)
    s=io.read()
    if not s then io.close(csvfile) else
      t={}; for x in s:gmatch("([^,]+)") do t[1+#t] = read(x) end
      return t end end end
```

```lua
-- __fmt(control:str, arg1,arg2...)__<br>sprintf emulation.
fmt = string.format
-- __fyi(x:str)__ <br> Print things in verbose mode.
fyi = function(...) if THE.verbose then print(...) end end
-- __lt(x:str):fun__ <br>Return a sort function on slot `x`.
function lt(x) return function(a,b) return a[x] < b[x] end end
-- __map(t:tab, f:fun):tab__ <br>Return a list, items filtered through `f`.
-- If `f` returns nil, then that item is rejected.
function map(t,f,  u) u={}; for k,v in pairs(t) do u[1+#u]=f(v) end return u end
-- __oo(i:tab)__ : <br>Pretty print `i`.
oo  = function(i) print(str(i)) end
-- __per(t:tab, p:float):float__
-- Return an item ,p-th way through `t`. `p=0.5` means return median.
function per(t,p) p=p*#t//1; return t[math.max(1,math.min(#t,p))] end
-- __push(t:tab, x:atom):x__ <br>Push `x` onto `t`, returning `x`.
function push(t,x) t[1+#t]=x; return x end
-- __rand(?x:num=1):num__<br> Generate a random number `1..x`.
rand= math.random
-- __rnd(n:num, places:int):num__ <p>Round `n` to `p` places.
function rnd(n, p)   local m=10^(p or 0); return math.floor(n*m+0.5)/m  end
-- __split(t, ?lo:float=1, ?j:float=#t, ?k:float=1):tab__
-- Return parts of `t` from `i` to `j` by steps `k`.
function splice( t, i, j, k,    u)
  u={}; for n=(i or 1)//1, (j or #t)//1, (k or 1)//1 do u[1+#u]=t[n] end return u
end

-- ## Demos

-- Place to store tests. To disable a test, rename `go.xx` to `no.xx`.
local go,no={},{}

function go.the() fyi(str(THE));  str(THE) return true end

function go.some( s)
  THE.some = 16
  s=SOME(); for i=1,10000 do s:add(i) end; oo(s:sorted())
  oo(s:sorted())
  return true end

function go.num( n)
  n=NUM(); for i=1,10000 do n:add(i) end; oo(n)
  return true end

function go.sym( s)
  s=SYM(); for i=1,10000 do s:add(math.random(10)) end;
  return s.has[9]==1045  end

function go.csv()
  for row in csv(THE.file) do oo(row) end; return true; end

function go.rows( rows)
  rows = ROWS(THE.file);
  map(rows.ys,print); return true; end

function go.mid(  r,bests,rests)
  r= ROWS(THE.file);
  bests,rests = r:bestRest()
  print("all",  str(r:mid(2)))
  print("best", str(r:clone(bests):mid(2)))
  print("rest", str(r:clone(rests):mid(2)))
  return true end

function go.range(  r,bests,rests)
  r= ROWS(THE.file);
  bests,rests = r:bestRest()
  for _,col in pairs(r.xs) do
    print("")
    for _,range in pairs(ranges(col, bests, rests)) do
       print(range, range.ys:score(1, #bests, #rests)) end  end
  return true end

function no.contrast(  r,bests,rests)
  r= ROWS(THE.file);
  bests,rests = r:bestRest()
  r:contrast(bests, rests)
  return true end

-- ## Starting up

-- Get a list of sorted demo names.
local going={}
for s,_ in pairs(go) do going[1+#going]=s end
table.sort(going)

-- Run the demos (or just `THE.go`
local fails=0
for _,s in pairs(go[THE.go] and {THE.go} or going) do
  for k,v in pairs(backup) do THE[k]=v end -- reset THE settings to the backup
  math.randomseed(THE.Seed)                -- reset the randomseed
  io.write(".")
  result = go[s]()
  if result ~= true then          -- report errors if demo does not return "true"
    fails = fails + 1
    print("--Error",s,status) end end

-- Check for rogue locals, then return the error counts (defaults to zero).
for k,v in pairs(_ENV) do  if not b4[k] then print("?",k,type(v)) end end
os.exit(fails)

-- ## Copyright <a name=copyright></a>
-- BSD 2-Clause License
--
-- 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 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
-- ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER 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.
```