```lua
local help = [[

l5 == a little lab of lots of LUA learning algorithms.
(c) 2022, Tim Menzies, BSD 2-clause license.

USAGE:
  lua l5.lua [OPTIONS]

OPTIONS:
  -cohen    F   Cohen's delta              = .35
  -data     N   data file                  = etc/data/auto93.csv
  -Dump         stack dump on assert fails = false
  -furthest F   far                        = .9
  -Format   s   format string              = %5.2f
  -keep     P   max kept items             = 512
  -p        P   distance coefficient       = 2
  -seed     P   set seed                   = 10019
  -todo     S   start up action (or 'all') = nothing
  -help        show help                  = false
  -want     F   recurse until rows^want    = .5

KEY: N=fileName F=float P=posint S=string

]]
-- NOTES: This code uses Aha's distance measure [^Ah91] (that can
-- handle numbers and symbols) to recursively divide data based on two
-- distant points (these two are found in linear time using the Fastmap
-- heuristic [Fa95]).
--
-- To avoid spurious outliers, this code use the 90% furthest points.
--
-- To avoid long runtimes, uses a subset of the data to learn where
-- to divide data (then all the data gets pushed down first halves).
--
-- To support explanation, optionally, at each level of recursion,
-- this code reports what ranges can best distinguish sibling clusters
-- C1,C2.  The  discretizer is inspired by the ChiMerge algorithm:
-- numerics are divided into, say, 16 bins. Then, while we can find
-- adjacent bins with the similar distributions in C1,C2, then
-- (a) merge then (b) look for other merges.
--
-- ## Namespace

-- Cache current globals, use at end to find rogue variables
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end

-- Defined local names.
local any,asserts,big,cli,csv,fails,firsts,fmt,goalp,ignorep,klassp
local lessp,map,main,many,max,merge,min,morep,new,nump,o,oo,per,pop,push
local r,rows,rnd,rnds,slots,sort,sum,thing,things,unpack

-- Classes have UPPER CASE names.
local CLUSTER, COLS, EGS,  EXPLAIN, NUM, ROWS = {},{},{},{},{},{}
local SKIP,    SOME, SPAN, SYM        = {},{},{},{}

-- ## Settings
-- Parse the help text for flags and defaults (e.g. -keep, 512).
-- Check for updates on those details from command line
-- (and and there,
-- some shortcuts are available;
-- e.g.  _-k N_ &rArr; `keep=N`;
-- and  _-booleanFlag_ &rArr; `booleanFlag=not default`).
local the={}
help:gsub("\n [-]([^%s]+)[^\n]*%s([^%s]+)",function(key,x)
  for n,flag in ipairs(arg) do
    if flag:sub(1,1)=="-" and key:find("^"..flag:sub(2)..".*") then
      x = x=="false" and true or x=="true" and "false" or arg[n+1] end end
  if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
    the[key] = tonumber(x) or x end end )

-- ---------------------------------------------------------------------------
-- this code reads csv files where the words on line1 define column types.
function ignorep(x) return x:find":$" end     -- columns to ignore
function klassp(x)  return x:find"!$" end     -- symbolic goals to achieve
function lessp(x)   return x:find"-$" end      -- number goals to minimize
function morep(x)   return x:find"+$" end      -- numeric goals to maximize
function nump(x)    return x:find"^[A-Z]" end -- numeric columns
function goalp(x)   return morep(x) or lessp(x) or klassp(x) end

-- strings
fmt = string.format

-- maths
big = math.huge
max = math.max
min = math.min
r   = math.random

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.Format) or "%s",x) end

-- tables
pop = table.remove
unpack = table.unpack
function any(t)        return t[r(#t)] end
function firsts(a,b)   return a[1] < b[1] end
function many(t,n, u)  u={}; for i=1,n do push(u,any(t)) end; return u end
function per(t,p)      return t[ (#t*(p or .5))//1 ] end
function push(t,x)     table.insert(t,x); return x end
function sort(t,f)     table.sort(t,f); return t end

-- meta
function map(t,f, u)  u={};for k,v in pairs(t) do push(u,f(v)) end; return u end
function sum(t,f, n)  n=0; for _,v in pairs(t) do n=n+f(v)     end; return n end
function slots(t, u)
  u={}
  for k,v in pairs(t) do k=tostring(k);if k:sub(1,1)~="_" then push(u,k) end end
  return sort(u) end

-- print tables, recursively
function oo(t) print(o(t)) end
function o(t)
  if type(t)~="table" then return tostring(t) end
  local key=function(k) return fmt(":%s %s",k,o(t[k])) end
  local u = #t>0 and map(t,o) or map(slots(t),key)
  return '{'..table.concat(u," ").."}" end

-- strings to things
function csv(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 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 push(t,thing(y)) end
  return t end

-- misc
function distance2Heaven(t,heaven,   num,d)
  for n,txt in pairs(heaven) do
    num = Num(at,txt)
    for _,z in pairs(t) do num:add(z.ys[n]) end
    for _,z in pairs(t) do z.ys[n] = num:distance2heaven(z.ys[n]) end end
  d = function(one) return (sum(one.ys)/#one.ys)^.5 end
  return sort(t, function(a,b) return d(a) < d(b) end) end
-- _____ _     _____ _____ _____ _____ _____
-- |     |     |___| [___  [___  |____ [___
-- |____ |____ |   | ____] ____] |____ ____]
--
function new(k,t) k.__index=k; k.__tostring=o; return setmetatable(t,k) end

-- COLS: turns list of column names into NUMs, SYMs, or SKIPs
function COLS.new(k,row,   i)
  i= new(k,{all={},x={},y={},names=row})
  for at,txt in ipairs(row) do  push(i.all, i:col(at,txt)) end
  return i end

function COLS.add(i,t)
  for _,col in pairs(i.all) do col:add( t[col.at] ) end
  return t end

function COLS.col(i,at,txt,     col)
  if ignorep(txt) then return SKIP:new(at,txt) end
  col = (nump(txt) and NUM or SYM):new(at,txt)
  push(goalp(txt) and i.y or i.x, col)
  if klassp(txt) then i.klass = col end
  return col end

-- NUM: summarizes a stream of numbers
function NUM.new(k,n,s)
  return new(k,{n=0,at=n or 0,txt=s or"",has=SOME:new(),ok=false,
                w=lessp(s or "") and -1 or 1, lo=big, hi=-big}) end

function NUM.add(i,x)
  if x ~= "?" then
    i.n = i.n + 1
    if i.has:add(x) then i.ok=false end
    i.lo,i.hi = min(x,i.lo), max(x,i.hi); end end

function NUM.dist(i,x,y)
  if     x=="?" and y=="?" then return 1
  elseif x=="?" then y=i:norm(y); x=y<0.5 and 1 or 0
  elseif y=="?" then x=i:norm(x); y=x<0.5 and 1 or 0
  else   x,y = i:norm(x), i:norm(y) end
  return math.abs(x-y) end

function NUM.distance2heaven(x, w)
  return ((i.w>0 and 1 or 0) - i:norm(x))^2 end

function NUM.mid(i) return per(i:sorted(), .5) end

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

function NUM.sorted(i)
  if i.ok==false then table.sort(i.has.all); i.ok=true end
  return i.has.all end

-- ROWS: manages `rows`, summarized in `cols` (columns).
function ROWS.new(k,inits,     i)
  i = new(k,{rows={},cols=nil})
  if type(inits)=="string" then for t in csv(inits) do i:add(t) end end
  if type(inits)=="table"  then for t in inits       do i:add(t) end end
  return i end

function ROWS.add(i,t)
  if i.cols then push(i.rows,i.cols:add(t)) else i.cols=COLS:new(t) end end

function ROWS.clone(i,  j) j= ROWS:new(); j:add(i.cols.names);return j end

function ROWS.dist(i,row1,row2,   d,fun)
  function fun(col) return col:dist(row1[col.at], row2[col.at])^the.p end
  return (sum(i.cols.x, fun)/ #i.cols.x)^(1/the.p) end

function ROWS.furthest(i,row1,rows,     fun)
  function fun(row2) return {i:dist(row1,row2), row2} end
  return unpack(per(sort(map(rows,fun),firsts), the.furthest)) end

function ROWS.half(i, top)
  local some, top,c,x,y,tmp,mid,lefts,rights,_
  some= many(i.rows, the.keep)
  top = top or i
  _,x = top:furthest(any(some), some)
  c,y = top:furthest(x,         some)
  tmp = sort(map(i.rows,function(r) return top:fastmap(r,x,y,c) end),firsts)
  mid = #i.rows//2
  lefts, rights = i:clone(), i:clone()
  for at,row in pairs(tmp) do (at <=mid and lefts or rights):add(row[2]) end
  return lefts,rights,x,y,c, tmp[mid] end

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

function ROWS.fastmap(i, r,x,y,c,     a,b)
  a,b = i:dist(r,x), i:dist(r,y); return {(a^2 + c^2 - b^2)/(2*c), r} end

-- SKIP: summarizes things we want to ignore (so does nothing)
function SKIP.new(k,n,s) return new(k,{n=0,at=at or 0,txt=s or""}) end
function SKIP.add(i,x)   return x end
function SKIP.mid(i)     return "?" end

-- SOME: keeps a random sample on the arriving data
function SOME.new(k,keep) return new(k,{n=0,all={}, keep=keep or the.keep}) end
function SOME.add(i,x)
  i.n = i.n+1
  if     #i.all < i.keep then push(i.all,x)          ; return i.all
  elseif r()     < i.keep/i.n then i.all[r(#i.all)]=x; return i.all end end

-- SYM: summarizes a stream of symbols
function SYM.new(k,n,s)
  return new(k,{n=0,at=n or 0,txt=s or"",has={},most=0}) 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 end

function SYM.dist(i,x,y) return(x=="?" and y=="?" and 1) or(x==y and 0 or 1) end
function SYM.mid(i)      return i.mode end
function SYM.div(i,   p)
  return sum(i.has,function(k) p=-i.has[k]/i.n;return -p*math.log(p,2) end) end

function SYM.merge(i,j,    k)
  k = SYM:new(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
  ei, ej, ejk= i:div(), j:div(), k:div()
  if i.n==0 or j.n==0 or .99*ek <= (i.n*ei + j.n*ej)/k.n then
    return k end end
```

```lua
--  ____ _    _  _ ____ ___ ____ ____
--  |    |    |  | [__   |  |___ |__/
--  |___ |___ |__| ___]  |  |___ |  \
--
-- CLUSTER: recursively divides data by clustering towards two distant points
function CLUSTER.new(k,egs,top)
  local i,want,left,right
  i      = new(k, {here=egs})
  top    = top or egs
  want = (#top.rows)^the.want
  if #egs.rows >= 2*want then
    left, right, i.x, i.y, i.c, i.mid = egs:half(top)
    if #left.rows < #egs.rows then
      i.left = CLUSTER:new(left,   top)
      i.right= CLUSTER:new(right, top) end end
  return i end

function CLUSTER.show(i,pre,  here)
  pre = pre or ""
  here=""
  if not i.left and not i.right then here= o(i.here:mid(i.here.cols.y)) end
  print(fmt("%6s : %-30s %s",#i.here.rows, pre, here))
  for _,kid in pairs{i.left, i.right} do
    if kid then kid:show(pre .. "|.. ") end end end

--  ____ _  _ ___  _    ____ _ _  _
--  |___  \/  |__] |    |__| | |\ |
--  |___ _/\_ |    |___ |  | | | \|

-- SPAN: keeps a random sample on the arriving data
function SPAN.new(k, col, lo, hi, has)
  return new(k,{col=col,lo=lo,hi=hi or lo,has=has or SYM:new()}) end

function SPAN.add(i,x,y,n) i.lo,i.hi=min(x,i.lo),max(x,i.hi); i.has:add(y,n) end
function SPAN.merge(i,j)
  local has = i.has:merge(j.has)
  if now then return SPAN:new(i.col, i.lo, j.hi, has) end end

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

function SPAN.score(i) return {i.has.n/i.col.n,  i.has:div()} end


-- EXPLAIN:
function EXPLAIN.new(k,egs,top)
  local i,top,want,left,right,spans,best,yes,no
  i    = new(k,{here = egs})
  top  = top or egs
  want = (#top.rows)^the.want
  if #top.rows >= 2*want then
    left,right = egs:half(top)
    spans  = {}
    for n,col in pairs(i.cols.x) do
      for _,s in pairs(col:spans(j.cols.x[n])) do
        push(spans,{ys=s:score(),it=s}) end end
    best   = distance2heaven(spans,{"+","-"})[1]
    yes,no = egs:clone(), egs:clone()
    for _,row in pairs(egs.rows) do
      (best:selects(row) and yes or no):add(row) end -- divide data in two
    if #yes.rows<#egs.rows then -- make kids if kid size different to parent siz
e
      if #yes.rows>=want then i.yes=EXPLAIN:new(yes,top) end
      if #no.rows >=want then i.no =EXPLAIN:new(no, top)  end end end
  return i end

function EXPLAIN.show(i,pre)
  pre = pre or ""
  if not pre then
    tmp = i.here:mid(i.here.y)
  print(fmt("%6s : %~30s %s", #i.here.rows, pre, o(i.here:mid(i.here.cols.y))))
  for _,pair in pairs{{true,i.yes},{false,i.no}} do
    status,kid = unpack(pair)
    k:shpw(pre .. "|.. ") end end end

function SYM.spans(i, j)
  local xys,all,one,last,xys,x,c n = {},{}
  for x,n in pairs(i.has) do push(xys, {x,"this",n}) end
  for x,n in pairs(j.has) do push(xys, {x,"that",n}) end
  for _,tmp in ipairs(sort(xys,firsts)) do
    x,c,n = unpack(tmp)
    if x ~= last then
      last = x
      one  = push(all, Span(i,x,x)) end
    one:add(x,y,n) end
  return all end

function NUM.spans(i, j)
  local xys,all,lo,hi,gap,xys,one,x,c,n = {},{}
  lo,hi = min(i.lo, j.lo), max(i.hi,j.hi)
  gap   = (hi - lo) / (6/the.cohen)
  for x,n in pairs(i.has) do push(xys, {x,"this",1}) end
  for x,n in pairs(j.has) do push(xys, {x,"that",1}) end
  one = Span:new(i,lo,lo)
  all = {one}
  for _,tmp in ipairs(sort(xys,first)) do
    x,c,n = unpack(tmp)
    if one.hi - one.lo > gap then one = push(all, Span(i, one.hi, x)) end
    one:add(x,y) end
  all          = merge(all)
  all[1   ].lo = -big
  all[#all].hi =  big
  return all end

function merge(b4,      j,n,now,a,b,merged)
  j,n,now = 0,#b4,{}
  while j < #b4 do
    j    = j+1
    a, b = b4[j], b4[j+1]
    if b then
      merged = a:merge(b)
      if merged then a,j = merged, j+1 end end
    push(now,a)
    j = j+1 end
  return #now == #b4 and b4 or merge(now) end
```

```lua
--  ___  ____ _  _ ____ ____
--  |  \ |___ |\/| |  | [__
--  |__/ |___ |  | |__| ___]
--
fails=0
function asserts(test, msg)
  print(test and "PASS: "or "FAIL: ",msg or "")
  if not test then
    fails=fails+1
    if the.dump then assert(test,msg) end end end

function EGS.nothing() return true end
function EGS.the()     oo(the) end
function EGS.rand()    print(r()) end
function EGS.some(s,t)
  s=SOME:new(100)
  for i=1,100000 do s:add(i) end
  for j,x in pairs(sort(s.all)) do
    --if (j % 10)==0 then print("") end
    --io.write(fmt("%6s",x))  end end
    fmt("%6s",x)  end end

function EGS.clone( r,s)
  r = ROWS:new(the.data)
  s = r:clone()
  for _,row in pairs(r.rows) do s:add(row) end
  asserts(r.cols.x[1].lo==s.cols.x[1].lo,"clone.lo")
  asserts(r.cols.x[1].hi==s.cols.x[1].hi,"clone.hi")
  end

function EGS.data( r)
  r = ROWS:new(the.data)
  asserts(r.cols.x[1].hi == 8, "data.columns") end

function EGS.dist( r,rows,n)
  r = ROWS:new(the.data)
  rows = r.rows
  n = NUM:new()
  for _,row in pairs(rows) do n:add(r:dist(row, rows[1])) end
  --oo(r.cols.x[2]:sorted()) end
  o(r.cols.x[2]:sorted()) end

function EGS.many(   t)
  t={}; for j=1,100 do push(t,j) end
  --print(oo(many(t, 10))) end
  o(many(t, 10)) end

function EGS.far(   r,c,row1,row2)
  r = ROWS:new(the.data)
  row1   = r.rows[1]
  c,row2 = r:far(r.rows[1], r.rows) end
  --print(c,"\n",o(row1),"\n", o(row2)) end

function EGS.half(   r,c,row1,row2)
  local lefts,rights,x,y,x
  r = ROWS:new(the.data)
  r:mid(r.cols.y)
  lefts,rights,x,y,c = r:half()
  lefts:mid(lefts.cols.y )
  rights:mid(rights.cols.y)
  asserts(true,"half") end

function EGS.cluster(r)
  r = ROWS:new(the.data)
  --CLUSTER:new(r):show() end
  CLUSTER:new(r) end

-- start-up
if arg[0] == "sl.lua" then
  if the.help then print(help:gsub("\nNOTES:*$","")) else
    local b4={}; for k,v in pairs(the) do b4[k]=v end
    for _,todo in pairs(the.todo=="all" and slots(EGS) or {the.todo}) do
      for k,v in pairs(b4) do the[k]=v end
      math.randomseed(the.seed)
      if type(EGS[todo])=="function" then EGS[todo]() end end
  end
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
  os.exit(fails)
else
  return {CLUSTER=CLUSTER, COLS=COLS, NUM=NUM, ROWS=ROWS,
          SKIP=SKIP, SOME=SOME, SYM=SYM,the=the,oo=oo,o=o}
end
-- git rid of SOME for rows
-- nss  = NUM | SYM | SKIP
-- COLS = all:[nss]+, x:[nss]*, y:[nss]*, klass;col?
-- ROWS = cols:COLS, rows:SOME
--
-- [^Ah91]: Aha, D.W., Kibler, D. & Albert, M.K. Instance-based   learning algor
ithms. Mach Learn 6, 37âM-^@M-^S66 (1991).  https://doi.org/10.1007/BF00153759
--
```