```lua
#!/usr/bin/env lua
-- vim: ts=2 sw=2 et:
-- (c) 2022, Tim Menzies
-- 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.
--------------------------------------------------------------------------------
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
local help = [[

gate: explore the world better, explore the world for good.
(c) 2022, Tim Menzies

      .-------.
      | Ba    | Bad <----.  planning= (better - bad)
      |    56 |          |  monitor = (bad - better)
      .-------.------.   |
              | Be   |   v
              |    4 | Better
              .------.

OPTIONS (inference control):
  -far   real  limit for search for far points    = .9
  -l     int   dist. co-efficient. Euclidean if 2 = 2
  -k     int   Bayes: handle rare classes         = 2
  -m     int   Bayes: handle rare values          = 1
  -min   real  min size                           = .5
  -seed  int   random number seed                 = 10019
  -keep  int   numbers to keep per column         = 512
  -wait  int   pre-learning, wait a few examples  = 5

OTHER:
  -h           show help                          = false
  -dump        enable stack dump on failures      = false
  -file        file with data                     = ../etc/data/auto93.csv
  -rnd   str   pretty print control for floats    = %5.3f
  -todo  str   start-up action                    = the

EXAMPLES:
  lua gate.lua -todo list    :   list all actions
  lua gate.lua -todo all     :   run all actions
]]

--------------------------------------------------------------------------------

-- define the local names
local the,go,no,fails = {}, {}, {}, 0
local abs,updates,cli,coerce,copy,csv ,demos,ent,fu,fmt,fmt2,gt,inc,last,log
local lt,map,map2,max,merge,merges,min,new,o,ok,obj,oo,ooo,per,push
local r,rnd,rnds,sd,settings,slots,sort,splice,sum

--                                                         ,:
--                                                       ,' |
--                                                      /   :
--                                                   --'   /
--                                                   \/ />/
--                                                   / /_\
--                                                __/   /
--                                                )'-. /
--                                                ./  :\
--                                                 /.' '
--                                               '/'
--                                               +
--                                              '
--                                            `.
--                                        .-"-
--                                       (    |
--                                    . .-'  '.
--                                   ( (.   )8:
--                               .'    / (_  )
--                                _. :(.   )8P  `
--                            .  (  `-' (  `.   .
--                             .  :  (   .a8a)
--                            /_`( "a `a. )"'
--                        (  (/  .  ' )=='
--                       (   (    )  .8"   +
--                         (`'8a.( _(   (
--                      ..-. `8P    ) `  )  +
--                    -'   (      -ab:  )
--                 '     _  `    (8P"Ya
--               _(     (    )b  -`.  ) +
--              ( 8)   ( _.aP" _a   \( \   *
--            +  )/     (8P   (88    )  )
--               (a:f    "     `"       `
--------------------------------------------------------------------------------
-- maths
r=    math.random
abs=  math.abs
log=  math.log
min=  math.min
max=  math.max
function ent(t,   n,e)
  n=0; for _,v in pairs(t) do n=n+v end
  e=0; for _,v in pairs(t) do e=e-v/n*log(v/n,2) end; return e end

function per(t,p)  return t[ ((p or .5)*#t) // 1 ] end

function sd(sorted,f,             ninety,ten)
  if #sorted <= 10 then return 0 end
  ninety,ten = per(sorted, .90), per(sorted, .10)
  if f then ninety,ten = f(ninety), f(ten) end
  return (ninety-ten)/2.564 end -- 2*(1.2 + 0.1*(0.9-0.88493)/(0.9032-0.88493))

-- lists
function last(t)      return t[#t] end
function inc(f,a,n)   f=f or{}; f[a]=(f[a] or 0) + (n or 1) return f end
function push(t,x)    t[1 + #t] = x; return x end
function sort(t,f)    table.sort(t,f); return t end
function map(t,f, u)  u={};for _,v in pairs(t)do u[1+#u]=f(v)  end;return u end
function map2(t,f, u) u={};for k,v in pairs(t)do u[k] = f(k,v) end;return u end

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

function slots(t,     u,public)
  function public(k) return tostring(k):sub(1,1) ~= "_" end
  u={};for k,v in pairs(t) do if public(k) then u[1+#u]=k end end
  return sort(u) end

function splice(t,lo,hi,   j,u)
  lo, hi = lo or 1, hi or #t
  u={}; for j=lo,hi do u[1+#u]=t[j] end; return u end

-- things to strings
fmt=  string.format
fmt2= function(k,v) return fmt(":%s %s",k,v) end

function ooo(t) print( #t>1 and o(t) or oo(t)) end
function o(t,s) return "{"..table.concat(map(t,tostring),s or",").."}" end
function oo(t,sep,    slot)
  function slot(k) return fmt2(k, t[k]) end
  return (t.is or"")..o(map(slots(t),slot),sep or" ") 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

-- strings to things
function coerce(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(src,      things)
  function things(s,  t)
    t={}; for y in s:gmatch("([^,]+)") do t[1+#t]=coerce(y) end; return t end
  src = io.input(src)
  return function(x) x=io.read()
    if x then return things(x) else io.close(src) end end end

-- misc
function fu(x) return function(t)   return t[x]        end end
function lt(x) return function(t,u) return t[x] < u[x] end end
function gt(x) return function(t,u) return t[x] > u[x] end end

function updates(obj,data)
  if   type(data) == "string"
  then for   row in csv(data)       do obj:update(row) end
  else for _,x in pairs(data or {}) do obj:update(x) end end
  return obj end

function merge(i,j,     k)
  k = i + j
  if k:div()*.95 <= (i.n*i:div() + j.n*j:div())/k.n then return k end end

function merges(b4,        a,b,c,j,n,tmp)
    j,n,tmp = 1,#b4,{}
    while j<=n do
      a, b = b4[j], b4[j+1]
      if b then
        c = merge(a,b)
        if c then a, j = c, j+1 end end
      tmp[#tmp+1] = a
      j = j+1 end
    return #tmp==#b4 and tmp or merges(tmp) end

-- startup, execution, unit tests
function settings(t,help)
  help:gsub("\n [-]([^%s]+)[%s]+[^\n]*%s([^%s]+)",function(k,x) t[k]=coerce(x) end)
  return t end

function cli(the,  flag)
  for k,v in pairs(the) do
    flag="-"..k
    for n,flag1 in ipairs(arg) do
      if flag1 == flag then
        v = v==false and"true" or v==true and"false" or arg[n+1]
        the[k] = coerce(v) end end end
  if the.h then os.exit(print(help)) else return the end end

function ok(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 demos(the,go,      demo1,defaults)
  function demo1(txt,f)
    assert(f, fmt("unknown start-up action: %s ",txt))
    the = copy(defaults)
    math.randomseed(the.seed or 10019)
    print(txt)
    f()
  end ---------------
  defaults = copy(the)
  if   the.todo=="all"
  then for _,txt in pairs(slots(go)) do
         demo1(txt,      go[txt]) end
  else   demo1(the.todo, go[the.todo])  end end

-- classes
function new(klass,...)
  local obj = setmetatable({},klass)
  local res = klass.new(obj,...)
  if res then obj = setmetatable(res,klass) end
  return obj end

function obj(name,    t)
  t={__tostring=oo, is=name or ""}; t.__index=t
  return setmetatable(t, {__call=new}) end
```

```lua
----------------------------------------------------------------------------
local Some,Sym,Num,Bin = obj"Some", obj"Sym", obj"Num", obj"Bin"
local Cols,Egs,Nb,Abcd = obj"Cols", obj"Egs", obj"Nb",  obj"Abcd"
local Cluster = obj"Cluster"
---------------------------------------------------------------------------
function Bin:new(at,name, lo,hi,ys)
  self.at, self.name          = at or 0, name or ""
  self.lo, self.hi, self.ys = lo, hi or lo, ys or Sym() end

function Bin:__tostring()
  local x,lo,hi,big = self.name, self.lo, self.hi, math.huge
  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 Bin:select(row)
  local x, lo, hi = row[self.at], self.lo, self.hi
  return x=="?" or lo == hi and lo == x or lo <= x and x < hi end

function Bin:update(x,y)
  if x<self.lo then self.lo = x end
  if x>self.hi then self.hi = x end
  self.ys:update(y) end

function Bin:div() return self.ys:div() end

function Bin:__add(other)
  return Bin(self.at, self.name, self.lo, after.hi, self.ys + other.ys) end
---------------------------------------------------------------------------
function Sym:new(at,name)
  self.at, self.name = at or 0, name or ""
  self.n, self.has, self.mode, self.most = 0,{},nil,0 end

function Sym:update(x,inc)
  if x ~= "?" then
    inc = inc or 1
    self.n = self.n + inc
    self.has[x] = inc + (self.has[x] or 0)
    if self.has[x] > self.most then self.most,self.mode = self.has[x],x end end
  return x end

function Sym:mid() return self.mode end
function Sym:div() return ent(self.has) end

function Sym:like(x,prior)
  return ((self.has[x] or 0) + the.m*prior)/(self.n + the.m) end

function Sym:dist(x,y)
  return x=="?" and y=="?" and 1 or x==y and 0 or 1 end

function Sym:__add(other,    out)
  out=Sym(self.at,self.name)
  for x,n in pairs(self.has) do out:update(x,n) end
  for x,n in pairs(other.has) do out:update(x,n) end
  return out end

function Sym:bins(other)
  local out = {}
  local function known(x) out[x] = out[x] or Bin(self.at, self.name, x,x) end
  for x,n in pairs(self.has)  do known(x); out[x].ys:update("left", n) end
  for x,n in pairs(other.has) do known(x); out[x].ys:update("right", n) end
  return map(slots(out), function(k) return out[k] end) end
---------------------------------------------------------------------------
function Some:new()
  self.kept, self.ok, self.n = {}, false,0 end

function Some:update(x,     a)
  self.n = 1 + self.n
  a      = self.kept
  if     #a  < the.keep        then self.ok=false; push(a,x)
  elseif r() < the.keep/self.n then self.ok=false; a[r(#a)]=x end end

function Some:has()
  if not self.ok then table.sort(self.kept) end
  self.ok = true
  return self.kept end
---------------------------------------------------------------------------
function Num:new(at,name)
  self.at, self.name = at or 0, name or ""
  self.w = self.name:find"-$" and -1 or 1
  self.some=Some()
  self.n,self.mu,self.m2,self.sd,self.lo,self.hi = 0,0,0,0,1E32,-1E32 end

function Num:update(x,_,   a,d)
  if x ~="?" then
    self.some:update(x)
    self.n  = self.n + 1
    self.lo = min(x, self.lo)
    self.hi = max(x, self.hi)
    d       = x - self.mu
    self.mu = self.mu + d/self.n
    self.m2 = self.m2 + d*(x - self.mu)
    self.sd = (self.m2<0 or self.n<2) and 0 or ((self.m2/(self.n - 1))^0.5) end
  return x end

function Num:__add(other,    out)
  out=Num(self.at,self.name)
  for _,x in pairs(self.some.kept) do out:update(x) end
  for _,x in pairs(other.some.kept) do out:update(x) end
  return out end

function Num:mid() return self.mu end
function Num:div() return self.sd end

function Num:like(x,_)
  local z, e, pi = 1E-64, math.exp(1), math.pi
  if x < self.mu - 4*self.sd then return 0 end
  if x > self.mu + 4*self.sd then return 0 end
  return e^(-(x - self.mu)^2 / (z + 2*self.sd^2))/(z + (pi*2*self.sd^2)^.5) end

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

function Num:norm(x,   lo,hi)
  lo,hi= self.lo, self.hi
  return x=="?" and x or hi-lo < 1E-9 and 0 or (x - lo)/(hi - lo) end

function Num:bins(other,         tmp,out,now,epsilon,minSize)
  tmp = {}
  for _,x in pairs(self.some.kept ) do push(tmp, {x=x, y="left"}) end
  for _,x in pairs(other.some.kept) do push(tmp, {x=x, y="right"}) end
  tmp = sort(tmp,lt"x") -- ascending on x
  out = {}
  now = push(out, Bin(self.at, self.name, tmp[1].x))
  epsilon = sd(tmp,fu"x") * the.cohen
  minSize = (#tmp)^the.leaves
  for j,xy in pairs(tmp) do
    if j > minSize and j + minSize < #tmp then -- leave enough for other bins
      if now.ys.n > minSize then               -- enough in this bins
        if xy.x ~= tmp[j+1].x then             -- there is a break in the data
          if now.hi - now.lo > epsilon then    -- "now" not trivially small
            now = push(out,  Bin(self.at, self.name, now.hi)) end end end end
    now:update(xy.x, xy.y) end
  out[1].lo    = -math.huge
  last(out).hi =  math.huge
  return merges(out) end
----------------------------------------------------------------------------
function Cols:new(names,    col)
  self.names, self.all, self.x, self.y, self.klass = names, {}, {}, {}, nil
  for at,name in pairs(names) do
    col = push(self.all, (name:find"^[A-Z]" and Num or Sym)(at,name))
    if not name:find":$"  then
      if name:find"!$" then self.klass=col end
      col.indep = not name:find"[-+!]$"
      push(col.indep and self.x or self.y, col) end end end

```

```
----------------------------------------------------------------------------
function Egs:new() self.rows, self.cols = {},nil end

function Egs:clone(data)
  return updates(Egs():update(self.cols.names), data) end

function Egs:update(row,   add)
  add = function(col) col:update(row[col.at]) end
  if   self.cols
  then map(self.cols.all,add); push(self.rows, row)
  else self.cols = Cols(row) end
  return self end

function Egs:mid(cols)
  return map(cols or self.cols.y, function(col) return col:mid() end) end

function Egs:div(cols)
  return map(cols or self.cols.y, function(col) return col:div() end) end

function Egs:like(row,egs,overall,       prior,like,col)
  prior = (#self.rows + the.k) / (overall + the.k * #egs)
  like  = log(prior)
  for at,x in pairs(row) do
    col = self.cols.all[at]
    if x ~= "?" and col.indep then like=like + log(col:like(x,prior)) end end
  return like end

function Egs:klass(row) return row[self.cols.klass.at] end

function Egs:better(row1,row2)
  local s1, s2, n, e = 0, 0, #self.cols.y, math.exp(1)
  for _,col in pairs(self.cols.y) do
    local a = col:norm(row1[col.at])
    local b = col:norm(row2[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 Egs:betters()
  return sort(self.rows, function(a,b) return self:better(a,b) end)  end

function Egs:dist(row1,row2,   d,n)
  d,n = 0, #self.cols.x
  for _,col in pairs(self.cols.x) do
    d = d + col:dist(row1[col.at], row2[col.at])^the.l end
  return (d/n)^(1/the.l) end

function Egs:around(row1, rows)
  function around(row2) return  {dist=self:dist(row1,row2),row=row2} end
  return sort(map(rows or self.rows,around), lt"dist") end

function Egs:far(row, rows)
  return per(self:around(row, rows or many(self.rows, the.some))).row end

function Eg:halves(top,   here)
  top = top or self
  here = Halved(eg,top)
  if here.lefts and #here.lefts.rows < #eg.rows then
    here.lefts  = here.lefts:halves(top)
    here.rights = here.rights:halves(top) end
  return here end

function Eg:bestsRests(     rests, keep, run, b4)
  function run(eg,b4,       here)
    here = Halved(eg, top, b4)
    if   here.lefts and #here.lefts.rows < #eg.rows
    then map(here.rights.rows,
             function(r) if r()<keep then rests:update(r) end end)
         return run(here.lefts, here.left)
    else return eg, rests end
  end -----------------------
  rests = self:clone()
  keep  = (#self.rows)^the.min
  keep  = the.keep*keep / (#self.rows - keep)
  b4    = self:far(any(self.rows))
  return run(self, b4) end

```

```
----------------------------------------------------------------------------
function Halved:new(eg,top,b4,     rows,some)
  self.top    = top or eg
  self.eg     = eg
  rows        = self.eg.rows
  if #eg.rows >= (#top.rows)^the.min then
    some      = many(rows, the.some)
    self.left = b4 or top:far( any(some), some)
    self.right=       top:far(self.left,  some)
    self.c    = self.top:dist(self.left, self.right)
    if b4 and eg:better(right,left) then
      self.left, self.right = self.right, self.left end
    self.lefts  = self.eg:clone()
    self.rights = self.eg:clone()
    for n,projection in pairs(self:projections(rows)) do
      (n < #rows//2 and self.lefts or self.rights):update(projection.row) end
    self.gaurd  = self.top:dist(left, last(left.rows)) end
  return self end

function Halved:projections(rows)
  return sort(map(rows, function(r) return self:project(r) end), lt"x") end

function Halved:project(row,   z,a,b,c)
  z   = 1/math.huge
  c,b,a = self.c, self.top:dist(row,self.right), self,top:dist(row,self.left)
  return {x  = (a^2 + c^2 - b^2) / (2*c + z),
          row = row} end
 ----------------------------------------------------------------------------
function Nb:new()
  self.all, self.some, self.log = nil, {}, {} end

function Nb:update(row)
  if   self.all
  then if #self.all.rows > the.wait then
           push(self.log, { want = self.all:klass(row),
                            got  = self:classify(row)   }) end
       self:train(row)
  else self.all = Egs():update(row) end end

function Nb:train(row,      k)
   k = self.all:klass(row)
   self.some[k] = self.some[k] or self.all:clone()
   self.some[k]:update(row)
   self.all:update(row)  end

function Nb:classify(row,   most,klass,tmp,out)
  most = -math.huge
  for klass,eg in pairs(self.some) do
    out = out or klass
    tmp = eg:like(row, self.some, #self.all.rows)
    if tmp > most then most,out = tmp, klass end end
  return out,most end
----------------------------------------------------------------------------
function Egs:tree(other,min,       kids,score)
  function gain(col1, col2, all,   sum,bins)
    sum = 0
    bins = col1:bins(col2)
    map(bins, function(bin)
                bin.here  = self
                bin.has = {self:clone(),self:clone()}
                sum = sum + bin.ys.n/all * bin.ys:div() end)
    return {bins=bins, gain=sum}
  end -----------------------
  n    = #self.rows + #other.rows
  stop = stop or n^the.min
  if   n < stop
  then return self
  else cols = map2(self.col.x, function(at,col)
                     return {w=gain(col, other.col.x[at], n), col=col} end)
       bins = sort(cols,fu"w")[1].bins
       for at,eg in pairs{self,other} do
         for _,row in pairs(eg.rows) do
           for _,bin in pairs(bins) do
              sub = bin.has[at]
              if bin:select(row) then sub:update(row); break end end end end
       self.kids = map(bins,
           function(bin) bin.kid = bin.has[1]:tree(bin.has[2]) end) end end
-- XXX not done yet. need to return the ocal kids
```

```
----------------------------------------------------------------------------
function Abcd:new(data,rx)
  self.data, self.rx = data or "", rx or ""
  self.yes, self.no  = 0,0
  self.known, self.a, self.b, self.c, self.d = {},{},{},{},{} end

function Abcd:exists(x,   new)
  new = not self.known[x]
  inc(self.known,x)
  if new then
    self.a[x]=self.yes + self.no; self.b[x]=0; self.c[x]=0; self.d[x]=0 end end

function Abcd:report(    p,out,a,b,c,d,pd,pf,pn,f,acc,g,prec)
  p = function (z) return math.floor(100*z + 0.5) end
  out= {}
  for x,xx in pairs( self.known ) do
    pd,pf,pn,prec,g,f,acc = 0,0,0,0,0,0,0
    a= (self.a[x] or 0); b= (self.b[x] or 0);
    c= (self.c[x] or 0); d= (self.d[x] or 0);
    if b+d > 0     then pd   = d     / (b+d)        end
    if a+c > 0     then pf   = c     / (a+c)        end
    if a+c > 0     then pn   = (b+d) / (a+c)        end
    if c+d > 0     then prec = d     / (c+d)        end
    if 1-pf+pd > 0 then g=2*(1-pf) * pd / (1-pf+pd) end
    if prec+pd > 0 then f=2*prec*pd / (prec + pd)   end
    if self.yes + self.no > 0 then
       acc= self.yes /(self.yes + self.no) end
    out[x] = {data=self.data,rx=self.rx,num=self.yes+self.no,
               a=a,b=b,c=c,d=d,acc=p(acc),
               prec=p(prec), pd=p(pd), pf=p(pf),f=p(f), g=p(g), class=x} end
  return out end

function Abcd:pretty(t,    s1,s2,d,s,u)
  print""
  s1  = "%10s | %10s | %4s | %4s | %4s | %4s "
  s2  = "| %3s | %3s| %3s | %4s | %3s | %3s |"
  d,s = "---", (s1 .. s2)
  print(fmt(s,"db","rx","a","b","c","d","acc","pd","pf","prec","f","g"))
  print(fmt(s,d,d,d,d,d,d,d,d,d,d,d,d))
  for x,u in pairs(sort(map(t,function(x) return x end),
                     function(a,b) return (a.b+a.d> b.b+b.d) end)) do
    print(fmt(s.." %s", u.data,u.rx,u.a, u.b, u.c, u.d,
                     u.acc, u.pd, u.pf, u.prec, u.f, u.g, u.class)) end end

function Abcd:adds(gotwants, show)
  for key,one in pairs(gotwants) do
    self:exists(one.want)
    self:exists(one.got)
    if one.want == one.got then self.yes=self.yes+1 else self.no=self.no+1 end
    for x,xx in pairs(self.known) do
      if   one.want == x
      then inc(one.want == one.got and self.d or self.b, x)
      else inc(one.got  == x       and self.c or self.a, x) end end end
  return show and self:pretty(self:report()) or self:report() end
```

```lua
--------------------------------------------------------------------------------
function go.list()
  map(slots(go), function(x) print(fmt("lua gate.lua –todo %s",x)) end) end

function go.the() ooo(the) end

function go.sort(   t)
  t={10,9,3}
  ooo(sort(t)) end

function go.ent() ok(abs(1.3788 - ent{a=4,b=2,c=1}) < 0.001,"enting") end

function go.ooo() ooo{cc=1,bb={ff=4,dd=5,bb=6}, aa=3} end

function go.copy(  t,u)
  t = {a=1,b=2,c={d=3,e=4,f={g=5,h=6}}}
  u = copy(t)
  t.c.f.g = 100
  ok(u.c.f.g ~= t.c.f.g, "deep copy") end

function go.rnds() ooo(rnds{3.421212, 10.1121, 9.1111, 3.44444}) end

function go.csv(  n)
  n=0; for row in csv(the.file) do n=n+1 end; ok(n==399,"stuff") end

function go.some(  s)
  the.keep = 64
  s = Some(); for i=1,10^6 do s:update(i) end
  ooo(s:has()) end

function go.num(     n,mu,sd)
  n, mu, sd = Num(), 10, 1
  for i=1,10^3 do
    n:update(mu + sd*math.sqrt(-2*math.log(r()))*math.cos(2*math.pi*r())) end
  ok(abs(n:mid() - mu) < 0.025, "sd")
  ok(abs(n:div() - sd) < 0.05,  "div")  end

function go.updates( n)
  print(updates(Num(),{1,2,3,4,5}) + updates(Num(),{11,12,13,14,15}))
  end

function go.sym(     s,mu,sd)
  s= Sym()
  for i=1,100 do
    for k,n in pairs{a=4,b=2,c=1} do s:update(k,n) end end
  ooo(s.has) end

function go.egs(f)
  for _,col in pairs(updates(Egs(),f or "../etc/data/diabetes.csv").cols.all) do
    print("\n",col) end end

function go.clone(f,     a,b)
  a = updates(Egs(),f or "../etc/data/diabetes.csv")
  b = a:clone(a.rows)
  print(a.cols.x[1].sd)
  print(b.cols.x[1].sd)
  ok(a.cols.x[1].sd == b.cols.x[1].sd, "same y") end

function go.abcd()
  local t={}
  for _ = 1,6 do push(t,{want="yes",got="yes"}) end
  for _ = 1,2 do push(t,{want="no",got="no"}) end
  for _ = 1,6 do push(t,{want="maybe",got="maybe"}) end
  for _ = 1,1 do push(t,{want="maybe", got="no"}) end
  Abcd():adds(t,true) end

function go.nb(f,    nb)
  nb = updates(Nb(),f or "../etc/data/diabetes.csv")
  Abcd():adds(nb.log, true) end

function go.nbsb()
  go.nb("../etc/data/soybean.csv") end

function go.bestrest(   eg,best,rest,rows,n)
  eg= updates(Egs(),"../etc/data/auto93.csv")
  rows = eg:betters()
  n    = (#rows)^.5 // 1
  best = splice(rows, 1,n)
  rest = splice(rows, #rows-n)
  best = eg:clone(best)
  rest = eg:clone(rest)
  ooo(rnds(best:mid()))
  ooo(rnds(rest:mid()))
end

--------------------------------------------------------------------------------
the = settings(the,help)

if   pcall(debug.getlocal, 4, 1)
then return {Num=Num, Sym=Sym, Egs=Egs} -- called as sub-module. return classes
else the = cli(the)  -- update `the` from command line
     demos(the,go)   -- run some demos
     for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
     os.exit(fails) end
```