```lua
-- vim: ts=2 sw=2 et:
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):
  -k     int   Bayes: handle rare classes          = 2
  -m     int   Bayes: handle rare values           = 1
  -seed  int   random number seed                  = 10019
  -keep  int   numbers to keep per column          = 512

OTHER:
  -h           show help                           = false
  -dump        enable stack dump on failures       = false
  -rnd   str   pretty print control for floats     = %5.3f
  -todo  str   start-up action ("all" == run all) = the ]]
------------------------------------------------------------------------------
local the,go,no,fails = {}, {}, {}, 0
local abs,adds,class,cli,coerce,copy,csv ,demos,ent,fmt,fmt2,log
local map,map2,max,min,o,ok ,oo,ooo,push,r,rnd,rnds,settings,slots,sort

-- 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

-- lists
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 u 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

-- 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

function class(name,    t,new)
  function new(klass,...)
    local obj = setmetatable({},klass)
    local res = klass.new(obj,...)
    if res then obj = setmetatable(res,klass) end
    return obj
  end --------
  t={__tostring=oo, is=name or ""}; t.__index=t
  return setmetatable(t, {__call=new}) end

function adds(obj,data)
  if   type(data)=="string"
  then for   row in csv(data)          do obj:add(row) end
  else for _,row in pairs(data or {}) do obj:add(row) end end
  return obj 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,fun)
    assert(fun, fmt("unknown start-up action: %s ",txt))
    the = copy(defaults)
    math.randomseed(the.seed or 10019)
    print(txt)
    fun()
  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
------------------------------------------------------------------------------
local Some=class("Some")
function Some:new()
  self.kept, self.ok, self.n = {}, false,0 end

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

------------------------------------------------------------------------------
local Num=class("Num")
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.sd,self.lo,self.hi = 0,0,0,1E32,-1E32 end

function Num:add(x,_,   a,d)
  if x ~="?" then
    self.some:add(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: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: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
------------------------------------------------------------------------------
local Sym=class("Sym")
function Sym:new(at,name)
  self.at, self.name = at or 0, name or ""
  self.has, self.mode, self.most = {},nil,0 end

function Sym:add(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

------------------------------------------------------------------------------
local Cols=class("Cols")
function Cols:new(names,    col)
  self.names = names
  self.all, self.x, self.y = {}, {}, {}
  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

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

function Egs:add(row,   add)
  add = function(col) col:add(row[col.at]) end
  if self.cols then push(self.rows, map(self.cols,add)) else
     self.cols = Cols(row) end 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,       n,prior,like,col)
  n=0; for _,eg in pairs(egs) do n = n + #eg.rows end
  prior = (#self.rows + the.k) / (n + 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: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 go.the() ooo(the) end

the = settings(the,help)

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