```
-- vim: filetype=lua ts=2 sw=2 et:
-- (c) 2022, Tim Menzies,  timm@ieee.org, opensource.org/licenses/Fair
-- 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.
--      __                                          __
--     /\ \__                                      /\ \
--     \ \ ,_\  __  __  __     __     _      \ \ \/'\
--      \ \ \/ /\ \/\ \/\ \  /'__`\  /'__`\    \ \ , <
--       \ \ \_\ \ \_/ \_/ \/\  __/ /\ \L\.\_  \ \ \\`\
--        \ \__\\ \___x___/'\ \____\\ \__/.\_\  \ \_\ \_\
--         \/__/ \/__//__/   \/____/ \/__/\/_/   \/_/\/_/

local help= [[
tweak: tries three weak learners for multi-objective optimization
(c) 2022, Tim Menzies,  timm@ieee.org, opensource.org/licenses/Fair

learner1: n times, discard half the data furthest from best
learner2: classify data according to presence of survivors of learner1
learner3: run learner1 on best "best" found by learner2

USAGE:
  alias twk="lua tweak.lua "
  twk [OPTIONS]

OPTIONS:
  --boot    -b  size of bootstrap            = 512
  --cohen   -c  cohen                        = .35
  --conf    -C  statistical confidence       = 0.05
  --cliffs  -l  cliff's delta                = 0.147
  --K       -K  manage low class counts      = 1
  --M       -M  manage low evidence counts   = 2
  --far     -F  how far to go for far        = .95
  --p       -p  coefficient on distance      = 2
  --seed    -S  seed                         = 10019
  --some    -s  sample size for distances    = 512
  --stop    -T  how far to go for far        = 20
  --min     -m  size of min space            = .5
  --best    -B  best percent                 = .05

OPTIONS (other):
  --dump    -d  dump stack+exit on error     = false
  --file    -f  file name                    = ../etc/data/auto93.csv
  --help    -h  show help                    = false
  --rnd     -r  rounding numbers             = %5.3f
  --go      -g  start up action              = nothing]] --[[

ABOUT THE CODE:
- Settings generated from "help" string
  - Settings can be updated from the strings seed in flags
  - Settings stored in the global "the"

- Layout code in chunks of size 120 lines (max), broken by line-feed
  - Chunk1=header; Chunk2=utils; Chunk3=objects; Chunk(last)=demos+start-up
- Layout lines 80 chars side (max)
  - So use 2 spaces for "tab"
- Do functions as one-liners (if possible)
- In order to define code in any order:
  - Near the top, define all function and Object names as "local"
- Otherwise, don't use the "local" keyword (too ugly)

- Minimize use of map (hard to debug)
- Object names are short and UPPER CASE
- Private object slots (that should not be printed) start with "_".
- Constructors need not return constructed instance.
- No inheritance (hard to debug)
- For code with many parameters, pass in a dictionary with named fields.

- Tests  in the "go" table at end. Reset settings to defaults after each.
- Tests check for error conditions using "ok" not "assert".
- Command line "-d -go x" crashed if test "x" fails, shows stack dump.
- Command line "-go x" calls test "go.x()"
- Command line "-go all" calls all tests.
- Command line "-h" shows help
- Command line "-S N" sets random seed (so "-S $RANDOM" is "full" random)

- 2nd last line: look for "rogue" globals (there should be none)
- Last line: exit to operating system with number of failures seen in tests

ABOUT THE CLASSES:
- "mean","mode" are generalized to "mid" (i.e. "mid-point")
- "standard deviation","entropy" are generalized  to "div" (i.e. "diversity")

- BIN holds the class labels seen between "lo" and "hi".
- EGS (examples) hold many ROWs, summarized in SYMbolic or NUMeric  columns.
- COLS is a factory for turning column names into NUMs or SYMs.
  - Numeric names start with upper case
  - Goal names ending with "-" or ""+ get weights -1,1 for minimize,maximize
  - Non-numeric class names end with "!"
  - Columns to be skipped have a name ending with ":"
  - Non-skipped columns are divided into COLS.y and COLS.x (for goal and other)
- "mid" and "div" for EGS are computed recursively by "mid,div" in NUMs,SYMs
- distances between two rows is computed recursively via "dist" in NUMs,SYMs

- ROW1 before ROW2 (i.e. ROW1<ROW2) if its goals dominate (using [CDOM])
- ROWs are recursively separated and clustered by [FASTMAP] random projections
- The distance between two ROWs (i.e. ROW1-ROW2) uses [AHA].
- To save space, ROWs are made once but can be passed around different EGS.
- ROWs have a "_data" pointer where it gets "lo,hi" info needed for distances.
  - For consistency, "_data" is set to the first EGS that holds that row.

REFERENCES:
- [AHA]    : Aha       : doi.org/10.1007/BF00153759
- [CDOM]   : Zitzler   : doi.org/10.1145/1830483.1830578
- [FASTMAP]: Faloutsos : doi.org/10.1145/568271.223812      --]]
---     _   _  ____  _   _      ____
---    | | | |  |  | |  |      [__
---    |_| |  |  | |  |___  ___]

local the,any,cells,copy,csv,fmt,fu,lt,many,map,normal = {}
local o,obj,ok,oo,per,push,R,rnd,rnds,shuffle,sort,slice,stats,string2thing
---                                        _
---     _   -|-  .-  o  .-   _   _     )   -|-  |-   o  .-   _   _
---    _>   |_   |   |  | |  (_|  _>   /_   |_  | |  |  | |  (_|  _>
---                             _|                                 _|
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

help:gsub("\n  ([-][-]([^%s]+))[%s]+(-[^%s]+)[^\n]*%s([^%s]+)",function(f1,k,f2,x)
  for n,flag in ipairs(arg) do if flag==f1 or flag==f2 then
    x = x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
  the[k] = string2thing(x) end)

function csv(src)
  src = io.input(src)
  return function(line, row)
    line=io.read()
    if not line then io.close(src) else
      row={}; for x in line:gmatch("([^,]+)") do row[1+#row]=string2thing(x) end
      return row end end end

function shuffle(t,    j)
  for i = #t, 2, -1 do
    j=math.random(i); t[i], t[j] = t[j], t[i]; end;
  return t end

---     ._ _   o   _   _
---     | | |  |  _>  (_
R = math.random
function fu(x)   return function(a)   return a[x] end end
function lt(x)   return function(a,b) return a[x] < b[x] end end
function normal(mu,sd)
  return mu + sd*math.sqrt(-2*math.log(R()))*math.cos(2*math.pi*R()) end

---     |  o   _  -|-   _
---     |  |  _>   |_  _>
function any(a, i)    i=R()*#a//1; i=math.max(1,math.min(i,#a)); return a[i] end
function many(a,n, u) u={}; for j=1,n do         u[1+#u]= any(a) end;return u end
function map(t,f, u)  u={}; for _,v in pairs(t) do u[1+#u]=f(v) end;return u end
function per(t,p, i)  i=(p or.5)*#t//1; return t[math.max(1,math.min(#t,i))] end
function push(t,x)    t[1+#t]=x; return x end
function sort(t,f)    table.sort(t,f) return t end
function slice(t,i,j,   u)
  u={}; for k=(i or 1), (j or #t) do u[1+#u] = t[k] 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

---     ._   .-  o  .-  -|-
---     |_)  |   |  | |  |_
---     |
fmt = string.format
function oo(t) print(o(t)) end
function o(t,    u,one,hide,sorted)
  if type(t) ~= "table" then return tostring(t) end
  sorted = #t>0 -- true when array's indexes are 1,2...#t
  hide= function(k) return tostring(k):sub(1,1) == "_" end
  one = function(k,v) return sorted and tostring(v) or fmt(":%s %s",k,v) end
  u={}; for k,v in pairs(t) do if not hide(k) then u[1+#u] = one(k,v) end end
  return (t.is or "").."{"..table.concat(sorted and u or sort(u)," ").."}" 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

--------------------------------------------------------------------------------
---
---    ____ ___  _ ____ ____ ___ ____
---    |  | |__] | |___ |     |  [__
---    |__| |__] _| |___ |___  |  ___]

local _id=0
local function id() _id=_id+1; return _id end

function obj(name,    t,new)
  function new(kl,...)
   local x=setmetatable({id=id()},kl); kl.new(x,...); return x end
  t = {__tostring=o, is=name or ""}; t.__index=t
  return setmetatable(t, {__call=new}) end

local SOME,SYM,BIN,NUM,COLS = obj"SOME",obj"SYM",obj"BIN",obj"NUM",obj"COLS"
local ROW,EGS,GO            = obj"ROW", obj"EGS", obj"GO"

---
---     _   _   ._ _    _
---    _>  (_)  | | |  (/_
function SOME:new() self.kept, self.ok, self.n = {}, false,0 end

function SOME:add(x,     a)
  self.n, a = 1 + self.n, self.kept
  if     #a  < the.some        then self.ok=false; push(a,x)
  elseif R() < the.some/self.n then self.ok=false; a[R(#a)]=x end end

function SOME:has()
  if not self.ok then sort(self.kept) end;self.ok=true; return self.kept end
---
---     _        ._ _
---    _>  \/  | | |
---        /
function SYM:new(pos,s)
  self.pos, self.txt= pos or 0,s or ""
  self.n, self.has, self.most, self.mode = 0,{},0,nil end

function SYM:sub(x,inc) return self:add(x, -(inc or 1)) end
function SYM:add(x,inc)
  if x ~= "?" then
    inc = inc or 1
    self.n = self.n + inc
    self.has[x] = (self.has[x] or 0) + inc
    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(   e)
  e=0; for _,m in pairs(self.has) do
         if m>0 then e = e-m/self.n * math.log(m/self.n,2) end end
  return e end

function SYM:__add(other,    out)
  out = SYM(self.pos,self.txt)
  for x,n in pairs(self.has) do out:add(x,n) end
  for x,n in pairs(other.has) do out:add(x,n) end
  return out end

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

function SYM:bins(rows,     out,known,x)
  out,known = {},{}
  for _,row in pairs(rows) do
    x = row.cells[self.pos]
    if x~="?" then
      known[x] = known[x] or push(out, BIN({txt=self.txt,  pos=self.pos,
                                             lo=x ,hi=x, ys=SYM()}))
      known[x].ys:add(row.klass) end end
  return out end

---
---     ._ |  |_|  ._ _ |
---     | |  |_|  | | |
function NUM:new(pos,s)
  self.pos, self.txt, self.lo, self.hi = pos or 0,s or "",1E32, -1E32
  self.n, self.some = 0,SOME()
  self.mu, self.m2, self.sd = 0,0,0
  self.w = self.txt:find"-$" and -1 or 1  end

function NUM:add(x,    _,d)
  if x ~="?" then
    self.some:add(x)
    self.n  = self.n + 1
    self.lo = math.min(x, self.lo)
    self.hi = math.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.n<2 or self.m2<0) and 0 or (self.m2/(self.n-1))^0.5 end
  return x end

-- does the has() sort
function NUM:mid() return per(self.some:has(),.5)  end
function NUM:div(  a)
  a=self.some:has()
  return #a<=10 and self.sd or (per(a,.9)-per(a,.1))/2.56 end

function NUM:same(x,y)
  print(math.abs(x-y), self:div()*the.cohen)
  return math.abs(x-y) < self:div()*the.cohen end

function NUM:merge(other,    out)
  out = NUM(self.pos,self.txt)
  for _,x in pairs(self.some.kept)  do out:add(x) end
  for _,x in pairs(other.some.kept) do out:add(x) end
  return out 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: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 math.abs(x - y) end

function NUM:bins(rows,      xy,div,xys,epsilon,small,b4,out)
  function xy(row, x)
    x=row.cells[self.pos]; if x~="?" then return {x=x,y=row.klass} end end
  function div(lo,hi,        x,y,cut,lhs,rhs,tmp,best,overall)
    lhs, rhs, overall = SYM(), SYM(), SYM()
    for i=lo,hi do overall:add( rhs:add(xys[i].y) ) end
    best = rhs:div()
    for i=lo,hi do
      x, y = xys[i].x, xys[i].y
      lhs:add( rhs:sub( y) )
      if lhs.n > small and rhs.n > small then
        if x ~= xys[i+1].x then
          if x - xys[lo].x > epsilon and xys[hi].x - x > epsilon then
            tmp = (lhs.n*lhs:div() + rhs.n*rhs:div())  / (lhs.n + rhs.n)
            if tmp < best then
              best,cut = tmp,i end end end end end
    if   cut
    then div(lo,    cut)
         div(cut+1, hi)
    else b4= push(out, BIN({txt=self.txt, pos=self.pos, lo=b4,
                             hi=xys[hi].x, ys=overall})).hi end
  end ------------------------------
  xys     = sort(map(rows,xy), lt"x")
  b4,out  = -math.huge, {}
  epsilon = (per(xys,.9).x - per(xys,.1).x) / 2.56*the.cohen
  small   = (#xys)^the.min
  div(1, #xys)
  out[#out].hi = math.huge
  return out end
```

```
---      _   _   |   _
---     (_  (_)  |  _>

function COLS:new(names,       it,num,sym,col)
  self.names, self.x, self.y, self.all = names, {},{},{}
  for pos,txt in pairs(names) do
    col = push(self.all, (txt:find"^[A-Z]" and NUM or SYM)(pos,txt))
    if not txt:find":$" then
      if txt:find"!$" then self.klass = col end
      push(txt:find"[-+!]$" and self.y or self.x, col) end end end

---     |_   o  ._
---     |_)  |  | |
function BIN:new(t)
  self.pos, self.txt  = t.pos, t.txt
  self.lo, self.hi, self.ys = t.lo, t.hi, t.ys or SYM() end

function BIN:__tostring()
  local x,lo,hi,big = self.txt, 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(t)
  t = t.cells and t.cells or t
  local x, lo, hi = t[self.pos], self.lo, self.hi
  return x=="?" or lo == hi and lo == x or lo <= x and x < hi end

function BIN:of(x) return self.ys.has[x] or 0 end

---     ._   _
---     |   (_)  \/\/
function ROW:new(data,t)
  self._data,self.cells, self.evaluated,self.rank,self.klass = data,t,false,0, fal
se end

function ROW:__sub(other,    cols,d,inc)
  d, cols = 0, self._data.cols.x
  for _,col in pairs(cols) do
    inc = col:dist(self.cells[col.pos], other.cells[col.pos])
    d   = d + inc^the.p end
  return (d / #cols) ^ (1/the.p) end

function ROW:__lt(other,   s1,s2,e,y,a,b)
  y= self._data.cols.y
  s1, s2, e = 0, 0,  math.exp(1)
  for _,col in pairs(y) do
     a = col:norm(self.cells[col.pos])
     b = col:norm(other.cells[col.pos])
     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:around(rows,   rowGap)
  function rowGap(row) return {row=row, gap=self - row} end
  return sort(map(rows or self.data.rows, rowGap), lt"gap") end

function ROW:far(rows) return per(self:around(rows), the.far).row end
---      _    _   _
---     (/_  (_|  _>
---           _|
function EGS:new()   self.rows,self.cols = {},nil end
function EGS:load(f) for t in csv(f) do self:add(t) end; return self end
function EGS:mid(t)  return map(t or self.cols.y,function(c)return c:mid()end)end
function EGS:div(t)  return map(t or self.cols.y,function(c)return c:div()end)end

function EGS:ranks(      any, all,first,now,n)
  for i,row in pairs(sort(self.rows)) do row.rank = (100*i/#self.rows)//1 end end

function EGS:evaluated(rows,  n)
  n=0;for _,row in pairs(rows or self.rows) do n=n+(row.evaluated and 1 or 0)end
  return n end

function EGS:add(t)
  if   self.cols
  then t = push(self.rows, t.cells and t or ROW(self,t)).cells
       for _,col in pairs(self.cols.all) do col:add(t[col.pos]) end
  else self.cols = COLS(t) end
  return self end

function EGS:clone(rows, out)
  out=EGS():add(self.cols.names)
  for _,row in pairs(rows or {}) do out:add(row) end
  return out end

function EGS:sway(rows,x,stop,rest,           rxs,some,y,c,best,mid)
  rows = rows or self.rows
  stop = stop or 2*the.best*#rows
  rest = rest or {}
  if #rows <= stop then return rows,rest,x end
  some = many(rows,the.some)
  x    = x or any(some):far(some)
  y    =      x:far(some)
  if y < x then x,y = y,x end
  c    = x - y
  x.evaluated = true
  y.evaluated = true
  rxs = map(rows,function(r) return {r=r, x=((r-x)^2+c^2-(r-y)^2)/(2*c)} end)
  best= {} -- things cloest to x or y, respectively
  for i,rx in pairs(sort(rxs, lt"x")) do
    push(i<=#rows*.5 and best or rest, rx.r) end
  return self:sway(best,x,stop,rest)  end

function EGS:rbins(rows,B,R,how,   v,bins,best,bests)
  function v(bin,  b,r)
    b = bin:of(true)  / (B+0.0001)
    r = bin:of(false) / (R+0.0001)
    return b^2/(b+r) end
  how = how or {}
  bins = {}
  for _,col in pairs(self.cols.x) do
    for _,bin in pairs(col:bins(rows)) do push(bins,bin) end end
  best = sort(bins,function(a,b) return v(a) > v(b) end)[1]
  bests= map(rows,function(row) if best:select(row) then return row end end)
  if  #bests < #rows
  then push(how,best)
       return self:rbins(bests,B,R,how )
  else return  rows,how end
end^K
-------------------------------------------------------------------------------
---    ___  ___  ___  ___  ___
---   |   \|__  |\/| |   ||__
---   |__/ |___ |  | |___| __]

function ok(test,msg)
  print("", test and "PASS "or "FAIL ", msg or "")
  if not test then
    GO.fails= GO.fails+1
    if the.dump then assert(test,msg) end end end

function GO:new(todo,    b4,go)
  b4={}; for k,v in pairs(the) do b4[k]=v end
  go={}; for k,_ in pairs(GO) do
           if k~="new" and type(GO[k])=="function" then go[1+#go]=k end end
  GO.fails = 0
  for _,x in pairs(todo=="all" and sort(go) or {todo}) do
    for k,v in pairs(b4) do the[k]=v end
    math.randomseed(the.seed)
    if GO[x] then print(x); GO[x]() end end end

function GO.rogue( t)
  t={}; for _,k in pairs{ "_G", "_VERSION", "arg", "assert", "collectgarbage",
  "coroutine", "debug", "dofile", "error", "getmetatable", "io", "ipairs",
  "load", "loadfile", "math", "next", "os", "package", "pairs", "pcall",
  "print", "rawequal", "rawget", "rawlen", "rawset", "require", "select",
  "setmetatable", "string", "table", "tonumber", "tostring", "type", "utf8",
  "warn", "xpcall"} do t[k]=true end
  for k,v in pairs(_ENV) do if not t[k] then print("?",k, type(v)) end end end

function GO.the() oo(the) end
function GO.eg( n,out)
  out =true
  n=0; for row in csv(the.file) do
         n=n+1; out=out and #row==8
                if n>1 then out=out and type(row[1])=="number" end end
  ok(out and n==399); end

function GO.some( s)
  s=SOME(); for i=1,10^6 do s:add(R(100)) end
  oo(s:has()) end

function GO.num(  n,s,t)
  local function sd(t,  n,d,m,m2)
    n,m,m2=0,0,0;for _,x in pairs(t) do n=n+1; d=x-m; m=m+d/n; m2=m2+d*(x-m) end
    return (m2/n)^0.5 end
  for i=1,5 do; print("")
    s=2; for r=1,6 do
      s=s*4
      t={}
      n=NUM(); for i=1,s do push(t, n:add(normal(10,2))) end
      print(fmt("%7.0f %6.2f %6.2f %6.2f",s,n:mid(),n:div(),sd(t))) end end end

function GO.rows( egs)
  egs=EGS():load(the.file)
  map(egs.cols.x,oo); print("");
  map(egs.cols.y,oo) end

function GO.dist( egs,    a,b,c,out)
  egs  = EGS():load(the.file)
  out = true
  for i=1,100 do
    a,b,c = any(egs.rows), any(egs.rows), any(egs.rows)
    out   = out and (b -a)==(a-b) and (a-a)==0 and ((a-b)+ (b-c) >= (a-c)) end
  ok(out,"dist") end

function GO.sort(    egs,rows,n)
  egs  = sort(EGS():load(the.file))
  rows= sort(egs.rows)
  n = .05*#rows//1
  print("what", o(map(egs.cols.y,function(c) return c.txt end)))
  print("all",  o(rnds(egs:mid())))
  print("best", o(rnds(egs:clone(slice(rows, 1, n)):mid())))
  print("rest", o(rnds(egs:clone(slice(rows, n+1 )):mid())))
  end

function GO.far(  egs,row2)
  egs = EGS():load(the.file)
  row2=egs:far(egs.rows[1])
  print(row2 - egs.rows[1])end

function GO.sway(  egs,best,rest)
  egs = EGS():load(the.file)
  best,rest = egs:sway()
  egs:ranks()
  for _,row in pairs(egs.rows) do if row.evaluated then oo(row.cells) end end
  print("all",  o(rnds(egs:mid())))
  print("rest", o(rnds(egs:clone(rest):mid())))
  print("best", o(rnds(egs:clone(best):mid())))
  oo(sort(map(best,function(row) return row.rank end))) end

function GO.symbins( egs)
  egs = EGS():load(the.file)
  for i,row in pairs(sort(egs.rows)) do row.klass = i<=#egs.rows//2 end
  map(egs.cols.x[4]:bins(egs.rows),oo) end

function GO.bins( egs)
  egs = EGS():load(the.file)
  for i,row in pairs(sort(egs.rows)) do row.klass = i<=.05*#egs.rows end
  for _,col in pairs(egs.cols.x) do
    print(fmt("\n%s",col.txt))
    map(col:bins(egs.rows),print) end end

function GO.shuffle(   t)
  t= {10,20,30,40,50,60,70,80,90}
  for i=1,20 do oo(shuffle(t)) end end

function GO.per()
  print(per({10,20,30,40},0.00)) end


function GO.rbins1() GO.rbins(1) end
function GO.rbins( max)
  max = max or 20

  local randoms1={[0]=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()}
  local randoms2={[0]=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()}
  local sways1={[0]=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()}
  local sways2={[0]=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()}
  local guesses={[0]=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()}
  local nguesses=NUM()
  local depths=NUM()
  local nsways1=NUM()
  local nsways2=NUM()
  local nbests1=NUM()
  local nbests2=NUM()
  for i=1,max do
    print("\n==========================")
    local egs = EGS():load(the.file)
    print("what", o(map(egs.cols.y,function(c) return c.txt end)))
    print("mid", o(egs:mid()))
    egs.rows = shuffle(egs.rows) --- <==== important!!!!
    local best1,rest1,top = egs:sway()
    print("best1s", o(egs:clone(best1):mid()))
    nsways1:add(egs:evaluated())
    nbests1:add(#best1)
    local best2,rest2,top = egs:sway(best1,top,10)
    print("best2s", o(egs:clone(best2):mid()))
    nsways2:add(egs:evaluated())
    nbests2:add(#best2)
    local anys1=many(egs.rows,1*#best2)
    local anys2=many(egs.rows,2*#best2)
    local klasses={}
    for _,row in pairs(best2)                            do push(klasses,row).klas
s=true end
    for _,row in pairs(slice(shuffle(rest1),1,4*#best2)) do push(klasses,row).klas
s=false end
    shuffle(klasses)
    local guess,how = egs:rbins(shuffle(klasses),#best2,4*#best2)
    print("guesses", o(egs:clone(guess):mid()))
    nguesses:add(#guess)
    depths:add(#how)
    print(""); map(how,print)
    egs:ranks()
    local anys1  = sort(map(anys1,function(row) return row.rank end))
    local anys2  = sort(map(anys2,function(row) return row.rank end))
    local best1 = sort(map(best1,function(row) return row.rank end))
    local best2 = sort(map(best2,function(row) return row.rank end))
    local guess = sort(map(guess,function(row) return row.rank end))
    for i,num in pairs(randoms1)  do num:add(per(anys1,i))  end
    for i,num in pairs(randoms2)  do num:add(per(anys2,i))  end
    for i,num in pairs(sways1)  do num:add(per(best1,i))  end
    for i,num in pairs(sways2)  do num:add(per(best2,i))  end
    for i,num in pairs(guesses)  do num:add(per(guess,i))  end
  end
  print(";")
  for _,kv in pairs({{rands1=randoms1},{rands2=randoms2},{sway1=sways1},{ sway2=sw
ays2},{guess=guesses}}) do
    for k,v in pairs(kv) do
      local t=map({0, .25, .5, .75},function(p) return fmt(",%5.1f",v[p]:mid()) end
 )
      print(";,",the.file,",",k,table.concat(t)) end  end

  print(";")
  print(";,",the.file,",nsways1,",nsways1:mid())
  print(";,",the.file,",nsways2,",nsways2:mid())
  print(";,",the.file,",nbests1,",nbests1:mid())
  print(";,",the.file,",nbests2,",nbests2:mid())
  print(";,",the.file,",nguesses,",nguesses:mid(),",",nguesses:div())
  print(";,",the.file,",depths,",depths:mid(),",",depths:div())
  end

function GO.ranks( egs)
  egs = EGS():load(the.file)
  egs:ranks()
  for _,row in pairs(egs.rows) do if row.rank>0 then print(row.rank,o(row.cells))
end end end

-------------------------------------------------------------------------------
---    ___ ___ ___ ___ ___
---   [__   |  |__| |__/  |
---   ___]  |  |  | |  \  |

if   pcall(debug.getlocal, 4, 1)
then return {the=the,any=any,any=csv,fmt=fmt,many=many,map=map,
             oo=oo,o=o,obj=obj,per=per,push=push,R=R,
             rnd=rnd,rnds=rnds,sort=sort,slice=slice,
             string2thing=string2thing,
             NUM=NUM, SYM=SYM}
else if the.help then print(help) else GO(the.go) end
     GO.rogue()
     os.exit(fails)  end
```