```lua
local your,our={},{}
our.b4={}; for k,_ in pairs(_ENV) do our.b4[k] = k end
our.help=[[

  -file   ../../data/auto93.csv
  -best   .5
  -help   false
  -dull   .35
  -rest   3
  -seed   10019
  -rnd    %.2f
  -task   -
  -p      2]]

local any,as,asserts,cells,copy,fmt,go,id,many, map,o,klass,push
local rand,randi,rnd,rows,same,slots,sort,thing,things
local COLS,EG,EGS,NUM,RANGE,SYM
-- ------------------------------------------------------------------
as    = setmetatable
fmt   = string.format
same = function(x,...) return x end

function many(t,n, u) u={};for j=1,n do push(u,any(t)) end; return u end
function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
function rand(lo,hi)
  your.seed = (16807 * your.seed) % 2147483647
  return (lo or 0) + ((hi or 1) - (lo or 0)) * your.seed / 2147483647 end

function push(t,x)  table.insert(t,x); return x end
function sort(t,f)  table.sort(t,f); return t end
function slots(t,u) u={};for x,_ in pairs(t) do u[1+#u]=x end;return sort(u) end
function map(t,f,  u)
  u={};for _,v in pairs(t) do u[1+#u]=(f or same)(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[k]=copy(v) end; return as(u,getmetatable(t)) end

function thing(x)
  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 t[1+#t]=thing(y) end; return t end

function rows(file,      x)
  file = io.input(file)
  return function()
    x=io.read()
    if x then
      x=x:gsub("%s+","");return things(x) else io.close(file) end end end

function rnd(x)
  return fmt(type(x)=="number" and x~=x//1 and your.rnd or"%s",x) end

function o(t,f,   u,key)
  key= function(k)
         if t[k] then return fmt(":%s %s", k, rnd((f or same)(t[k]))) end end
  u = #t>0 and map(map(t,f),rnd) or map(slots(t),key)
  return "{"..table.concat(u, " ")..'}' end

function asserts(test,msg)
  msg=msg or ""
  if test then return print("PASS : "..msg) end
  our.fails = our.fails+1
  print("FAIL : "..msg)
  if your.Debug then assert(test,msg) end end

function id() our.id = 1+(our.id or 0); return our.id end

function klass(t)
  t.__index=t; return as(t,{__call=function(_,...) return t.new(...) end}) end
-- ------------------------------------------------------------------
SYM=klass{}
function SYM.new(at,s)
  return as({at=at or 0,txt=s or "",has={},n=0,most=0,mode=nil},SYM) end

function SYM.__tostring(i)
  return fmt("SYM{:at %s :txt %s :mode %s :has %s}",
             i.at, i.txt, i.mode, o(i.has)) end

function SYM.add(i,x)
  if x ~= "?" then
    i.n = i.n+1
    i.has[x] = 1 + (i.has[x] or 0) end
  return x end

function SYM.mid(i,     most,out)
  most=-1
  for x,n in pairs(i.has) do if n>most then out,most=x,n end end; return out end

function SYM.ranges(i,j,bests,rests)
  local tmp,out,x = {},{}
  for _,pair in pairs{{bests,"bests"}, {rests,"rests"}} do
    for _,row in pairs(pair[1]) do
      x = row.has[i.at]
      if x~= "?"  then
         tmp[x] = tmp[x] or SYM()
         tmp[x]:add(pair[2]) end end end
  for x,stats in pairs(tmp) do push(out, RANGE(i,x,x,stats)) end
  return out end
-- ------------------------------------------------------------------
NUM=klass{}
function NUM.new(at,s, big)
  big = math.huge
  return as({lo=big, hi=-big, at=at or 0, txt=s or "",
             n=0, mu=0, m2=0, sd=0,
             w=(s or "):find"-" and -1 or 1},NUM) end

function NUM.__tostring(i)
  return fmt("NUM{:at %s :txt %s :lo %s :hi %s :mu %s :sd %s}",
              i.at, i.txt,  i.lo, i.hi, rnd(i.mu), rnd(i:div())) end

function NUM.mid(i) return i.mu end
function NUM.div(i) return i.n <2 and 0 or (i.m2/(i.n-1))^0.5 end

function NUM.add(i,x,     d)
  if x~="?" then
    i.n = i.n+1
    d  = x - i.mu
    i.mu = i.mu + d/i.n
    i.m2 = i.m2 + d*(x-i.mu)
    i.lo = math.min(x,i.lo); i.hi = math.max(x,i.hi) end
  return x end

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

function NUM.ranges(i,j, bests,rests)
  local ranges,x,lo,hi,gap,tmp = {}
  hi  = math.max(i.hi, j.hi)
  lo  = math.min(i.lo, j.lo)
  gap = (hi - lo)/your.bins
  tmp = lo
  for j=lo,hi,goal do push(ranges,RANGE(i, tmp, tmp+gap)); tmp= tmp+gap end
  ranges[1].lo       = -math.huge
  ranges[#ranges].hi = math.huge
  for _,pair in pairs{{bests,"bests"}, {rests,"rests"}} do
    for _,row in pairs(pair[1]) do
      x = row.has[i.at]
      if x~= "?"  then
        ranges[(x - lo)//gap].stats:add(pair[2]) end end end end

-- ------------------------------------------------------------------
RANGE=klass{}
function RANGE.new(col,lo,hi,stats)
  return as({col=col, lo=lo, hi=hi or lo, stats=stats}, RANGE) end

function RANGE.__tosting(i)
  return fmt("RANGE{:col %s :lo %s :hi %s :stats %s}",
             col.txt, i.lo, i.hi, o(i.stats)) end

-- ------------------------------------------------------------------
COLS=klass{}
function COLS.new(t,      i,where,now)
  i = as({all={}, x={}, y={}},COLS)
  for at,s in pairs(t) do
    now = push(i.all, (s:find"^[A-Z]" and NUM or SYM)(at,s))
    if not s:find":" then
      push((s:find"-" or s:find"+") and i.y or i.x, now) end end
  return i end

function COLS.__tostring(i, txt)
  function txt(c) return c.txt end
  return fmt("COLS{:all %s :x %s :y %s", o(i.all,txt), o(i.x,txt), o(i.y,txt)) end

EGS=klass{}
function EGS.clone(i,inits,     j)
  j = EGS()
  j:add(map(i.cols.all, function(col) return col.txt end))
  for _,x in pairs(inits or {}) do  j:add(x) end
  return j end

function COLS.add(i,t,       add)
  t = t.has and t.has or t
  function add(col,  x) x=t[col.at]; if x~="?" then col:add(x) end; return x end
  return map(i.all, add) end

function COLS.better(i,row1,row2)
  local s1,s2,e,n,a,b = 0,0,10,#i.y
  for _,col in pairs(i.y) do
    a  = col:norm(row1.has[col.at])
    b  = col:norm(row2.has[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

-- ------------------------------------------------------------------
EG=klass{}
function EG.new(t)         return as({has=t, id=id()},EG) end
function EG.__tostring(i) return fmt("EG%s%s", i.id,o(i.has)) end
-- ------------------------------------------------------------------
EGS=klass{}
function EGS.new()          return as({rows={}, cols=nil},EGS) end
function EGS.__tostring(i) return fmt("EGS{#rows %s :cols %s", #i.rows,i.cols) end
function EGS.file(i,f)      for row in rows(f) do i:add(row) end; return i end

function EGS.clone(i,inits,     j)
  j = EGS()
  j:add(map(i.cols.all, function(col) return col.txt end))
  for _,x in pairs(inits or {}) do  j:add(x) end
  return j end

function EGS.add(i,row)
  row = row.has and row.has or row
  if i.cols then push(i.rows, EG(i.cols:add(row))) else i.cols=COLS(row) end end


function EGS.mid(cols)
  return map(cols or i.cols.all,
             function(col) print(col); return col:mid() end) end

function EGS.bestRest(i)
  local best,rest,tmp,bests,restsFraction = {},{},{}
  i.rows = sort(i.rows, function(a,b) return i.cols:better(a,b) end)
  bests  = (#i.rows)^your.best
  restsFraction = (bests * your.rest)/(#i.rows - bests)
  for j,x in pairs(i.rows) do
     if      j       <= bests          then push(best,x)
     elseif rand() <  restsFraction then push(rest,x) end end
  return best,rest end
-- ------------------------------------------------------------------
RANGE=klass{}
function RANGE.new(col,lo,hi,stats)
  return as({col=col, lo=lo, hi=hi or lo, ys=stats or SYM(),all={}},RANGE) end

function RANGE.__tostring(i)
  return fmt("RANGE{:col %s :lo %s :hi %s :ys %s}",i.col,i.lo,i.hi,o(i.ys)) end
-- ------------------------------------------------------------------
our.go, our.no = {},{}; go=our.go
function go.settings() print("our",o(our)); print("your",o(your)) end

function go.sample() print(EGS():file(your.file)) end

function go.clone()
  a= EGS():file(your.file); print(a)
  b= a:clone()  end

function go.sort(   i,a,b)
  i= EGS():file(your.file)
  a,b=i:bestRest()
  print(#a, #b)
end
-- ------------------------------------------------------------------
our.help:gsub("\n  [-]([^%s]+)[^\n]*%s([^%s]+)", function(slot, x)
  for n,flag in ipairs(arg) do
    if   flag:sub(1,1)=="-" and slot:match("^"..flag:sub(2)..":*")
    then x = x=="false" and "true" or x=="true" and "false" or arg[n+1] end end
  your[slot] = thing(x) end)

if your.help then print(our.help) end

our.defaults=copy(your)
our.failures=0
for _,x in pairs(our.task=="all" and slots(our.go) or {your.task}) do
  if type(our.go[x]) == "function" then our.go[x]() else print("?", x) end
  your = copy(our.defaults)
end
for k,v in pairs(_ENV) do if not our.b4[k] then print("?",k,type(v)) end end
os.exit(our.failures)
```