```
                 ___                     ___
            /\__ \     /\___\          /\__ \
            \//\ \    \//\ \          \//\ \          __  __     __
   __  __     \ \ \      \ \ \           \ \ \    /\ \/\ \  /'__`\
  /'__`\ \     \_\ \_     \_\ \_ ___      \_\ \_  \ \ \_\ \/\ \L\.\_
 /\ \L\.\_   /\____\   /\____\/\_\   /\____\ \____\/\ \__/.\_\
 \ \__/.\_\   \/____/   \/____/\/_/   \/____/ \/____/  \/__/\/_/
  \/__/\/_/
```

```
-- ## About XAI and b(Ai)ttery
-- Explainable AI (XAI) is a subset of AI that tries
-- to build models that people can read and understand and critique and easily change.
-- B(Ai)ttery
-- is a small set of classes that implements a few interesting
-- XAI tools.
--
-- For the "big picture" on XAI, see the Vilone and Logno' 2020 systematic review.
--
-- For a small set of really useful XAI tactics, see below.
--
-- ### About XAI
-- For years
-- I used   XAI to _augment_ other more opaque AI tools.
-- But that meant I was not explaining the real inference
-- process, just some frail copy of what was really going on.
--
-- Then I found that XAI can (sometimes) actually
-- _replace_ other AI tools since, at least in the domains
-- I've explored, XAI tools can make better conclusions,
-- faster, and those conclusions are explicable to people.
--
-- (Not always of course. If you gave me 10,000 wavelets
-- from a signal processing package then of course I'd
-- reach for a deep learner.
-- But if you wanted to _tune_ that deep
-- learner, then I'd still use this code since it
-- just runs a few what-of queries on the
-- most important parts of the data.)
--
-- XAI should be designed with an understanding of human
-- cognitive processes. People are clever, as Davenport and Beck remind us,
-- they  have [fixed and limited attention spans
-- which they  hoard and use sparingly.
-- Herbert Simon say that humans  use heuristic "short cuts" that let
-- them satisfy the demands of their work, just enough
-- before rushing off to their next
-- task.
--
-- Once such short-cut is the "cue"; i.e. a small range
-- of some variable that most effects the outcome. Feature
-- extraction and weighting  is the process of finding
-- those cues. This code can be summarized as "the hunt
-- for 'cues'".
--
-- Another short-cut is sampling; i.e. don't look at
-- everything, just a few things. There are many ways to
-- sample and this code exploits them all (random,
-- reservoir, extreme).
--
-- ### References
-- - Thomas H. Davenport and John C. Beck. (2001).
--   [The Attention economy](https://ubiquity.acm.org/article.cfm?id=376626).
--   Ubiquity 2001, May (May 1 - May 31, 2001),
-- - Gigerenzer, G. (2008).
--   [Why Heuristics Work](https://pure.mpg.de/rest/items/item_2100099/component/file_2100098/content).
--   Perspectives on Psychological Science, 3(1), 20âM-^@M-^S29.
-- - Vilone, Giulia & Longo, Luca. (2020).
--   [Explainable Artificial Intelligence: a Systematic Review](https://arxiv.org/pdf/2006.00093.pdf)
-- - Simon, Herbert A. (1956).
--   [Rational Choice and the Structure of the Environment](https://uk.sagepub.com/sites/default/files/upm-binaries/25239_Chater~Vol_1~Ch_03.pdf)
--   Psychological Review. 63 (2): 129âM-^@M-^S138.
all=require"lib"
all.the = all.opts( [[
BAITTERY: semi-supervised multi-objective optimization XAI
(c) 2022 Tim Menzies <timm@ieee.org> BSD2 license

From N items, find and explain the best ones, using just log(N) evals.
PASS1 (guess): eval two distant items on multi-objective criteria.
       Prune everything nearest the worst one. Recurse on rest.
PASS2 (guess again): do it again, using better items from first pass.
PASS3 (explain): recursively discretize attributes on how well they
       distinguish the best and worst items (seen in second pass).

USAGE:
  lua go.lua [OPTIONS]

OPTIONS:
  -M  --Min    min size of space                    =  .5
  -b  --bins   max number of bins                   =  16
  -F  --Far    how far to look for remove points    =  .95
  -k  --k      Bayes hack: low attribute frequency  =  2
  -m  --m      Bayes hack: low class frequency      =  1
  -p  --p      distance coefficient (2=Euclidean)   =  2
  -s  --seed   random number seed                   =  10019
  -S  --Some   max number of nums to keep           =  256
  -w  --wait   wait this number before testing      =  10

OPTIONS (other):
  -f  --file   file           =  ../../data/auto93.csv
  -g  --go     start-up goal  =  nothing
  -h  --help   show help      =  false ]])

return all
-- XX
-- This code contains
-- B(Ai)TTERY (a set of AI-related classes) and
--  various AI tools, coded on top of B(Ai)TTERY.
--
-- One of the  idea here is that that there the thing we call "data
-- mining" shares many of its internal data structures and algorithms
-- with the thing we call "optimization". So once we build those
-- internal things, then building "data miners" or "optimizers"
-- is a  pretty trivial extension.
--
-- ### Apps
--  Naive Bays Classifier
--
-- Trees (regression and decision)
--
--  Recursive random projections
--
-- SHORTR:
-- Semi-supervised multi-objective optimization XAI
-- (from N items, find and explain the best ones, using just log(N) evals).
-- PASS1 (guess): eval two distant items on multi-objective criteria.
--        Prune everything nearest the worst one. Recurse on rest.
-- PASS2 (guess again): do it again, using better items from first pass.
-- PASS3 (explain): recursively discretize attributes on how well they
--       distinguish the best and worst items (seen in second pass).
--
-- ### Coding conventions
-- Before reading this, it might  be best to
-- review these [local coding conventions](https://github.com/timm/shortr/blob/master/CONTRIBUTE.md).
-- ## Why this code?
--  This code is an experiment in "less-is-more". Death to mash-ups and their associated
--  problems with technical debt and security problems that leak in from all
--  the parts used in the assembly.
--
-- <b>Tony Hoare:</b><br>
-- <em>"Inside every large program is a small program struggling to get out."</em><p>
-- <b>Alan Perlis:</b><br><em>"Simplicity does not precede complexity, but follows it."</em><p>
-- <b>Dieter Rams:</b><br><em>"Less, but better."</em>
--
-- Now that you've done _it_, did you really understand _it_? Let's check.
--
-- Can you do _it_ better?
-- Can you now
-- write _it_ in fewer lines and do you know how to make _it_ run faster?
-- Can you see how _it_ is same/different to other things?
-- And can you use those similarities to do more things with  _it_?
-- Finally, can you teach _it_ quickly to newcomers?
--
-- E.g. do I understand a multi-objective semi-supervised explanation algorithms?
-- Well, Let's check.
--
-- Here's all that, most of which is coded in B(Ai)TTERY
-- that could be used for other learners.
--
-- Also included here is literate programming,
-- self-documenting code and support for test-driven development.
-- All in around 500 lines of LUA: <br>
--
-- `awk '!/^(--|[ \t]*$)/{n++}`
-- `END {print n" lines"}' *.lua`
-- => 500 lines
--
-- Share and enjoy.
--
--
-- ### Role Models
-- People that inspire me to code less, but better:<br>
-- [Jack Diederich](https://www.youtube.com/watch?v=o9pEzgHorH0), [Hilary Mason](https://www.youtube.com/watch?v=12btv0yUPNQ),
-- [Brian McFee](https://brianmcfee.net/papers/ismir2011_sptree.pdf),
-- [Brian Kernighan](https://www.oreilly.com/library/view/beautiful-code/9780596510046/ch01.html),
-- [Joel Grus](https://github.com/joelgrus/data-science-from-scratch).<p>
-- Especially the LISPers: <br>
-- ([Peter Seibel](https://gigamonkeys.com/book/)
--    ([Conrad Barski](https://doc.lagout.org/programmation/Lisp/Land%20of%20Lisp_%20Learn%20to%20Program%20in%20Lisp%2C%20One%20Game%20at%20a%20Time%20%5BBarski%202010-11-15%5D.pdf)
--    ([Paul Graham](http://www.paulgraham.com/onlisp.html)<br>
--      ([Peter Norvig](http://norvig.com/lispy.html)
--        ([Guy Steele](https://dspace.mit.edu/bitstream/handle/1721.1/5790/AIM-353.pdf?sequence=2&isAllowed=y)))))).
```

```
                 ___                     ___
            /\__ \     /\___\          /\__ \
            \//\ \    \//\ \          \//\ \          __  __     __
   __  __     \ \ \      \ \ \           \ \ \    /\ \/\ \  /'__`\
  /'__`\ \     \_\ \_     \_\ \_ ___      \_\ \_  \ \ \_\ \/\ \L\.\_
 /\ \L\.\_   /\____\   /\____\/\_\   /\____\ \____\/\ \__/.\_\
 \ \__/.\_\   \/____/   \/____/\/_/   \/____/ \/____/  \/__/\/_/


-- ## class COLS:  make NUMs or SYMs

local all=require"all"
local obj, push = all.obj, all.push
local NUM, SYM = require"NUM", require"SYM"

--> COLS(names:[str]) :COLS -> Factory. Turns a list of names into NUMs or SYMs.
-- Goal columns get added to `i.y` and others to `i.x` (unless denoted `ignored`).
-- A klass column goes to `i.klass`.
local COLS = obj("COLS", function(i,names)
  i.names, i.x, i.y, i.all,i.klass, i.names = names, {}, {},  {}
  for at,txt in pairs(names) do
    local col = (txt:find"^[A-Z]" and NUM or SYM)(at,txt)
    push(i.all, col)
    if not col.txt:find":$" then
      push(col.txt:find"[!+-]$" and i.y or i.x, col)
      if col.txt:find"!$" then i.klass=col end end end end )

--> add(i:COLS: row:ROW) -> Update columns using data from `row`.
function COLS.add(i,row)
  for _,cols in pairs{i.x,i.y} do
    for _,col in pairs(cols) do col:add(row.cells[col.at]) end end end

return COLS
```

```
                                  ___
                                 /\__ \
                                 \//\ \
  __     __      __  __     __    __
 /'__`\ /'__`\   /\ \/\ \  /'__`\ /'__`\
/\ \L\ \/\ \L\ \  \ \ \_\ \/\ \L\ /\ \L\.\_
\ \____ \ \____/\ \ \____/\ \____\ \__/.\_\
 \/___L\ \/___/  \ \/___/  \/____/\/__/\/_/
   /\____/
   \_/__/

-- ## Test suite.
local all = require"all"
local chat,cli,csv,maps,on = all.chat, all.cli, all.csv, all.maps, all.on
local settings,sort,the    = all.settings, all.sort, all.the

local COLS,NUM, ROWS = require"COLS", require"NUM", require"ROWS"
local SOME, SYM      = require"SOME", require"SYM"

-- To disable a test, rename it from `go` to `no`.
local go,no = {},{}

-- Print `the`.
function go.THE() chat(the); return true end

-- Sort some numbers.
function go.SORT() chat(sort{10,5,1,15,0}); return true end

-- Iterate over 2 lists
function go.MAPS()
  chat(maps({1,2,3},{10,20,30},
        function(x,y) return x+y end)); return true end

--  Summarize stream of numbers
function go.NUMS()
  local n=NUM(); for i=1,1000 do n:add(i) end; chat(n)
  return true end

-- Keep a sample of 32 nums (out of 1000).
function go.SOME()
  local s=SOME{32}; for i=1,1000 do s:add(i) end
  chat(sort(s.kept)); return true end

--  Summarize stream of symbols
function go.SYM()
  local s=SYM()
  for i=1,1000 do for _,c in pairs{"a","a","b"} do s:add(c) end end
  chat(sort(s.kept)); return true end

-- Print CSV file.
function go.CSV() csv(the.file, chat); return true end

-- Try initializing some columns from a list of names.
function go.COLS() chat(COLS{"aa","Bb","Cc-"}.x); return true end

-- Load data from a csv file to a ROWS object.
function go.ROWS( rs)
  rs=ROWS():fill(the.file)
  chat(rs.cols.x[1])
  chat(rs.cols.y); return true end

-- Print klass names
function go.KLASS()
  local file = "../../data/diabetes.csv"
  local s=SYM()
  for _,row in pairs(ROWS():fill(file).rows) do s:add(row:klass()) end
  chat(s.kept)
  return true end

-- Load data from a csv file to a ROWS object.
function go.BETTERS( rs)
  rs=ROWS():fill(the.file)
  sort(rs.rows) end
-------
-- ### Start
the = cli(the)
on(the, go)
```

```
 ___                  ___                  ___
/\__ \      __      /\__ \          /\__ \
\//\ \    /\_\    \ \ \____\        \//\ \     __   __    __
  \ \ \   /\ \ \   \ \ \ ,__ \'\       \ \ \   /\ \/\ \  /'__`\
   \_\ \_  \ \ \    \ \ \ \L\ \ \___     \_\ \_ \ \ \_\ \ \ \L\.\_
   /\____\  \ \_\    \ \_,__/\\_\    /\____\ \ \____/\ \ \__/.\_\
   \/____/   \/_/     \/___/ \/_/    \/____/  \/___/  \/__/\/_/



-- ## Library Functions
local lib={}
-- ### Linting

--> rogues() -> Find rogue locals. Run `rogues()` _last_ after everything else.
local b4={}; for k,v in pairs(_ENV) do b4[k]=k end
function lib.rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end
-- ### Meta

--> lt(x:str):function -> Returns a function that sorts on `x`
function lib.lt(x) return function(a,b) return a[x] < b[x] end end

--> same(x:any):any -> Return x, unchanged.
lib.same=function(x) return x end

-- ### Maths

--> R(max:?num=1):num -> return a random number `0..max`.
lib.R = math.random

--> rnd(x:num, places:int):num -> return `x` rounded to some number of `places`.
function lib.rnd(x, places)
  local mult = 10^(places or 2)
  return math.floor(x * mult + 0.5) / mult end

--> rnds(t:num, places:?int=2):t -> return items in `t` rounds to `places`.
function lib.rnds(t, places)
  local u={};for k,x in pairs(t) do u[k] = rnd(x,places or 2) end; return u end
-- ###  Lists

--> splice(t:tab,start=?int=1,stop:?num=#t,step:?num=1):t -> pull items
-- `start` to `stop`, stepping by `step`.
function lib.splice(t, start, stop, step)
  local u={}
  for n=(start or 1)//1,(stop or #t)//1,(step or 1)//1 do u[1+#u]=t[n] end
  return u end

--> sort(t:tab, f:fun) :tab -> Return `t`, sorted of function `f` (default "<").
function lib.sort(t,f) table.sort(t,f); return t end
--> push(t:tab, x:any) :x -> Add `x` to end of `t`; return `t`.
function lib.push(t,x) t[1+#t] = x; return x end
--> per(t:tab, p:?float=.5) :x -> Return `p`-th ranked item from `t`.
function lib.per(t,p) p=p*#t//1; return t[math.max(1,math.min(#t,p))] end
--> map(t:tab, f:fun): tab ->
--> kap(t:tab, f:fun): tab ->
--> maps(list1:tab, list2:tab, f:fun): tab ->
--> kaps(list1:tab, list2:tab, f:fun): tab -> Return items in `t`, filtered thru `f`.
-- If `f` returns nil, then the output table shrinks. `kap` and `kaps` pass the
-- key and value to `f`. `maps` and `kaps` pass items from two lists.
function lib.map(t,f,     u) u={};for _,x in pairs(t) do u[1+#u]=f(x) end;return u end
function lib.kap(t,f,     u) u={};for k,x in pairs(t) do u[1+#u]=f(k,x) end;return u e
nd
function lib.maps(t,u,f,  v) v={};for k,x in pairs(t) do v[1+#v]=f(x,u[k]) end;return
v end
function lib.kaps(t,u,f,  v) v={};for k,x in pairs(t) do v[1+#v]=f(k,x,u[k]) end;retur
n v end
-- ### String to thing

--> thing(s:str):any -> Coerce string to whatever
-- is simplest (boolean or integer or float or, if all else fails, a string).
function lib.thing(x)
  x = x:match"^%s*(.-)%s*$"
  if x=="true" then return true elseif x=="false" then return false else
    return math.tointeger(x) or tonumber(x) or x end  end

--> words(s:str, sep:str, fun:fun):tab -> Return `t` filled with `s`, split  on `sep`.
function lib.words(s,sep,fun,       t)
   fun = fun or lib.same
   t={};for x in s:gmatch(lib.fmt("([^%s]+)",sep)) do t[1+#t]=fun(x) end; return t end

--> csv(file:str,  fun:fun):tab -> Call `fun` with lines, split on ",".
function lib.csv(file, fun)
  local file = io.input(file)
  while true do
    local line = io.read()
    if not line then return io.close(file) else
      fun(lib.words(line, ",", lib.thing)) end end end
-- ### Thing to string

--> fmt(s:str,...) :str -> emulate prinft
lib.fmt=string.format

--> cat(t:tab):str -> Return table as string. For key-indexed lists, show keys (sorted
).
function lib.cat(t,     key,u)
  function key(k,v) if (tostring(k)):sub(1,1)~="_" then return lib.fmt(":%s %s",k,v) en
d end
  u=  #t>1 and lib.map(t,f or tostring) or lib.sort(lib.kap(t,key))
  return (t._is or "")..("..table.concat(u," ")..")" end

--> chat(t:tab):t -> Print table (as string). Return `t`.
function lib.chat(t) print(lib.cat(t)); return t end
-- ### Settings

--> opts(x:str) :tab -> Parse `str` for lines with `--`; then pull keys+defaults.
function lib.opts(x)
  local t = {}
  x:gsub("\n ([-][^%s]+)[%s]+([-][-]([^%s]+))[^\n]*%s([^%s]+)",
           function(f1,f2,k,x) t[k] = lib.thing(x) end)
  t._HELP = x
  return t end

--> cli(t:tab) :tab -> For keys in `t`, look for updates on command-line.
-- Things with boolean defaults are flipped via `--flag`.
-- Other keys need `--flag value`.  Print the help
-- (if `-h` appears on command line). Return a table with setting `key`s and
-- `value`s. IMPORTANT NOTE: this function alters-in-place the table `t`
-- that is passed in-- which means that it alters settings for anything pointing
-- to `t`.
function lib.cli(t)
  for key,x in pairs(t) do
    x = tostring(x)
    local long, short = "--"..key, "-"..key:sub(1,1)
    for n,flag in ipairs(arg) do
```

```
      if flag==short or flag==long then
        x = x=="false" and "true" or x=="true" and "false" or arg[n+1]
        t[key] = lib.thing(x) end end end
  if t.help then os.exit(print(t._HELP:gsub("[%u][%u%d]+","\27[1;32m%1\27[0m"),"")) end
  return t end
-- ### Tests

--> on(opts:tab, tests:[fun]) -> Run some tests.
-- If  `opt.go=="all"`, then run all tests, sorted on their name.
-- Before each test, reset random seed and the options `opts.
function lib.on(opts,tests)
  local fails, old = 0, {}
  for k,v in pairs(opts) do old[k]=v end
  local t=opts.go=="all" and lib.kap(tests,function(k,_) return k end) or {opts.go}
  for _,txt in pairs(lib.sort(t)) do
    local fun = tests[txt]
    if type(fun)=="function" then
      for k,v in pairs(old) do opts[k]=v end -- reset opts to default
      math.randomseed(opts.seed or 10019)    -- reset seed to default
      print(">> ",txt)
      local out = fun()
      if out ~= true then fails=fails+1
                          print(lib.fmt("FAIL: [%s] %s",txt,out or "")) end end end
  lib.rogues()
  os.exit(fails) end -- if fails==0 then our return code to the OS will be zero.
-- ### Objects

--> obj(name:str, fun:fun):object -> Return a klass `name` with constructor `fun`.
-- Add a unique `id` and a `tosting` method (that uses `cat` (above).
local _id = 0
function lib.obj(name,fun,    t,new,x)
  function new(kl,...) _id=_id+1; x=setmetatable({_id=_id},kl);fun(x,...); return x en
d
  t = {__tostring=lib.cat,_is=name}; t.__index=t
  return setmetatable(t, {__call=new}) end
-----
-- ### Return

return lib
```

```
 __                  ___
/\ \                /\_ \
\ \ \____     ___   \//\ \      __     __     __
 \ \ '__`\  /'___`\   \ \ \    /'__`\ /'__`\ /'__`\
  \ \ \L\ \/\ \L\ \ \_\ \_ \_ /\  __/ /\  __/ /\ \L\.\_
   \ \_,__/\ \____/ /\____\ \_\\ \____\ \____\ \__/.\_\
    \/___/  \/___/  \/____/\/_/ \/____/\/____/\/__/\/_/
```

```
-- ## class NB: classifier
local all=require"all"
local obj,push,the = all.obj, all.push, all.the

local NB = obj("NB", function (i,src,report)
  i.overall, i.dict, i.list = nil, {},{}
  report = report or print
  Data.ROWS(src, function(row)
    if not i.overall then i.overall = ROWS(row)  else -- (0) eat row1
      row = i.overall:summarize(row)  -- XX add to overall
      if #i.overall.rows > the.wait then report(Row.klass(row), NB.guess(i,row)) end
      NB.train(i,row) end end)              -- add tp rows's klass
  end)

function NB.train(i,row)
  local kl = row:klass()
  i.dict[kl]     = i.dict[kl] or push(i.list,i.overall.clone()) --klass is known
  i.dict[kl].txt = kl                       -- each klass knows its name
  i.dict[k]:add(row) end                    -- update klass with row

function NB.keymax(i,row)
  local most,out = -1, nil
  for key,rows in pairs(i.dict) do
    tmp = rows:like(row, #i.list,#i.overall.rows)
    if tmp > most then most,out = tmp,key end end
  return key end

return NB
```

```
 __                  ___
/\ \                /\_ \
\ \ \____     ___   \//\ \      __     __     __
 \ \ '__`\  /'___`\   \ \ \    /'__`\ /'__`\ /'__`\
  \ \ \L\ \/\ \L\ \ \_\ \_ \_ /\  __/ /\  __/ /\ \L\.\_
   \ \_,__/\ \____/ /\____\ \_\\ \____\ \____\ \__/.\_\
    \/___/  \/___/  \/____/\/_/ \/____/\/____/\/__/\/_/
```

```
-- ## class NUM: summarize numbers
local all = require"all"
local obj,push,the = all.obj, all.push, all.the
local SOME = require"some"

--> NUM(at:?int, txt:?str) :NUM -> Summarize a stream of numbers.
local NUM = obj("NUM", function(i,at,txt)
  i.at, i.txt, i.n, i.kept =  at or 0, txt or "", 0, SOME(the.Some)
  i.w =  i.txt:find"-$"  end)

--> add(i:NUM: x:num, n:?int=1) -> `n` times,update `i`'s SOME object.
function NUM.add(i,x,n)
  if x ~="?" then
    for _ = 1,(n or 1) do i.n=i.n+1; i.kept:add(x) end end end

--> clone(i:(SYM|NUM)) :(SYM|NUM) -> Return a class of the same structure.
function NUM.clone(i) return NUM(i.at, i.txt) end

--> div(i:NUM):tab -> Return `div`ersity of a column
-- (its tendency _not_ to be a its central tendency). To understand this code
-- recall &pm;1 to &pm;2 sds covers 66 to 95% of the Gaussian prob. In between,
-- at &pm;1.28, we cover 90%. So (p90-p10)/(2*1.28) returns one sd.
function NUM.div(i)
  local a=i.kept:has(); return (per(a,.9) - per(a,.1))/2.56 end

--> like(i:NUM, x:any) -> Return the likelihood that `x` belongs to `i`.
function NUM.like(i,x,...)
  local sd,mu=i:div(), i:mid()
  if sd==0 then return x==mu and 1 or 1/big end
  return math.exp(-.5*((x - mu)/sd)^2) / (sd*((2*math.pi)^0.5)) end

--> mid(i:NUM)):tab -> Return a columns' `mid`ddle
function NUM.mid(i)
  local a=i.kept:has(); return per(a,.5) end

--> norm(i:NUM, x:num):num -> Normalize `x` 0..1 for lo..hi,
function NUM.norm(i,x)
  local a=i.kept:has(); return (a[#a]-a[1])<1E-9 or (x-a[1])/(a[#a]-a[1]) end

return NUM
```

```
 __                  ___
/\ \                /\_ \
\ \ \____     ___   \//\ \      __     __     __
 \ \ '__`\  /'___`\   \ \ \    /'__`\ /'__`\ /'__`\
  \ \ \L\ \/\ \L\ \ \_\ \_ \_ /\  __/ /\  __/ /\ \L\.\_
   \ \_,__/\ \____/ /\____\ \_\\ \____\ \____\ \__/.\_\
    \/___/  \/___/  \/____/\/_/ \/____/\/____/\/__/\/_/
```

```
-- ## class ROW:hold 1 record
-- See also [ROWS](rows.html) that holds multiple records.
-- And [NUM](num.html) and [SYM](sym.html) that summarize the
-- columns of the records.
local all = require"all"
local lt,map,obj,sort = all.lt, all.map, all.obj, all.sort

--> ROW(of:ROWS, cells:tab) :ROW -> Place to store one record
-- (and stats on how it is used; e.g. `i.evaled=true` if we touch the y values.
local ROW = obj("ROW", function(i,of,cells)
  i._of,i.cells,i.evaled = of,cells,false end)

--> i:ROW - j:ROW -> return distance between `i` and `j`
function ROW.__lt(i,j)
  local d, cols = 0, i._of.cols.x
  for _,col in pairs(cols) do
    local inc = col:dist(i.cells[col.at], j.cells[col.at])
    d         = d + inc^the.p end
  return (d / #cols) ^ (1/the.p) end

--> around(i:ROW, rows:?[ROW]):tab ->  return rows in this table
-- sorted by distance to `i`. `rows` defaults to the rows of this ROWS.
function ROW.around(i, rows)
  local function rowGap(j) return {row=j, gap=i - j} end
  return sort(map(rows or i._of.rows, rowGap), lt"gap") end

--> better(i:ROW, j:ROW):boolean -> should `i` proceed before `j`?
function ROW.better(i,j)
  i.evaled, j.evaled = true, true
  local s1, s2, ys = 0, 0, i._of.cols.y
  for _,col in pairs(ys) do
    local x,y =  i.cells[c.at], j.cells[c.at]
    x,y = col:norm(x), col:norm(y)
    s1  = s1 - 2.7183^(col.w * (x-y)/#ys)
    s2  = s2 - 2.7183^(col.w * (y-x)/#ys) end
  return s1/#ys < s2/#ys  end

--> far(i:ROW,rows:?[ROW]):ROW -> find something `far` away.
function ROW.far(i,rows) return per(Row.around(i,rows), the.Far).row end

--> klass(i:ROW):any -> Return the class value of this record.
function ROW.klass(i) return i.cells[i._of.cols.klass.at] end

return ROW
```

```
                                                  ___
                                                 /\_ \
 _ __   ___     __   __   __     ____           \//\ \      __  __     __
/\`'__\ / __`\  /\ \ /\ \ /\ \   /',__\            \ \ \    /\ \/\ \  /'__`\
\ \ \/ /\ \L\ \ \\ \ \_/ \_/ \/\__,  `\            \_\ \_   \ \ \_\ \ /\ \L\.\_
 \ \_\ \ \____/ \ \___x___/'\/\____/\/\_\           /\____\  \ \____/ \ \__/.\_\
  \/_/  \/___/   \/__//__/   \/___/ \/_/           \/____/   \/___/   \/__/\/_/
```

```
-- ## class ROWS: store many ROW
local all = require"all"
local csv,map,obj       = all.csv, all.map,  all.obj
local push,rnd,rnds,the = all.push, all.rnd, all.rnds, all.the
local COLS,ROW          = require"COLS",require"ROW"

--> ROWS(names:?[str], rows:?[ROW]) :ROWS -> Place to store many ROWS
--  and summarize them (in `i.cols`).
local ROWS = obj("ROWS", function(i,names,rows)
  i.rows, i.cols = {}, (names and COLS(names) or nil)
  for _,row in pairs(rows or {}) do i:summarize(row) end end)

--> add(i:ROWS: row:ROW) -> add ROW to ROWS, update the summaries in `i.cols`.
function ROWS.add(i,t)
  t = t.cells and t or ROW(i,t)
  if i.cols then i.cols:add(push(i.rows, t)) else i.cols=COLS(t.cells) end end

--> ROWS.clone(init:?[ROW]) :ROWS -> Return a ROWS with same structure as `i`.
-- Optionally, `init`ialize it with some rows. Add a pointer back to the
-- original table that spawned `eve`rything else (useful for some distance calcs).
function ROWS.clone(i,init)
  local j=ROWS(i.cols.names,init)
  return j end

--> fill(i:ROWS: src:(str|tab)):ROWS -> copy the data from `src` into `i`.
function ROWS.fill(i,src)
  local what2do = type(src)=="table" and map or csv
  what2do(src, function(t) i:add(t) end)
  return i end

--> like(i:ROWS,row;ROW,nklasses:num,nrows:num):num -> Return
-- P(H)*&prod;<sub>i</sub> (P(E<sub>i</sub>|H)). Do it with logs
-- to handle very small numbers.
function ROWS.like(i,row, nklasses, nrows)
  local prior,like,inc,x
  prior = (#i.rows + the.k) / (nrows + the.k * nklasses)
  like  = math.log(prior)
  row   = row.cells and row.cells or row
  for _,col in pairs(i.cols.x) do
    x = row[col.at]
    if x and x ~= "?" then
      inc  = col:like(x,prior)
      like = like + math.log(inc) end end
  return like end

--> mids(i:ROW,p:?int=2,cols=?[COL]=i.cols.y):tab -> Return `mid` of columnss
-- rounded to `p` places.
function ROWS.mids(i,p,cols,    t)
  t={}
  for _,col in pairs(cols or i.cols.y) do t[col.txt]=col:mid(p) end
  return rnds(t,p or 2) end

return ROWS
```

```
                                                  ___
                                                 /\_ \
  ____    ___     ___ ___      __    /'__`\        \ \ \    /\ \/\ \  /'__`\
 /',__\  /  __`\ /' __` __`\  /'__`\ /\__, `\        \_\ \_  \ \ \_\ \ /\ \L\.\_
/\__, `\/\ \L\ \/\ \/\ \/\ \/\  __/ \/\____/\/\_\      /\____\  \ \____/ \ \__/.\_\
\/\____/\ \____/\ \_\ \_\ \_\\ \____\ \/___/ \/_/      \/____/   \/___/   \/__/\/_/
 \/___/  \/___/  \/_/\/_/\/_/ \/____/\/_/                \/____/  \/___/  \/__/\/_/
```

```
-- ## class SOME: keep some nums
local all=require"all"
local obj,push,R,sort,the= all.obj, all.push, all.R, all.sort, all.the

--> SOME(max:?int) :SOME -> collect, at most, `max` numbers.
local SOME = obj("SOME", function(i,max)
  i.kept, i.ok, i.max, i.n = {}, true, max, 0  end)

--> add(i:SOME: x:num)-> `n` times,update `i`.
-- Helper function for NUM. If full then at odds `i.some/i.x`, keep `x`
-- (replacing some older item, at random).
function SOME.add(i,x)
  if x ~= "?" then
    i.n = i.n + 1
    if #i.kept < i.max     then i.ok=false; push(i.kept,x)
    elseif R() < i.max/i.n then i.ok=false; i.kept[R(#i.kept)]=x end end end

--> has(i:SOME):tab -> Ensure contents are sorted. Return those contents.
function SOME.has(i)
  i.kept = i.ok and i.kept or sort(i.kept); i.ok=true; return i.kept ; end

return SOME
```

```
                                                  ___
                                                 /\_ \
       __      __                                \//\ \      __  __     __
      /\ \ __ /\ \                                 \ \ \    /\ \/\ \  /'__`\
 ____ \ \ \ /.\ \    __      ____                  \_\ \_  \ \ \_\ \ /\ \L\.\_
/',__\  \ \ \ \/\ \  /'__`\   /',__\                /\____\  \ \____/ \ \__/.\_\
/\__, `\  \ \ \\ \_/ /\ \L\.\_ /\__, `\ __          \/____/   \/___/   \/__/\/_/
\/\____/   \ \_\\\ \__/.\_\ \__/\____/\/\_\
 \/___/     \/__/ \/__/\/_/  \/__/ \/___/ \/_/
```

```
-- ## Summarize data
local the=require"the"
local obj,per = _.obj,_.per

--- ## Adding
function ROWS.add(i,row)
  i.cols:add( push(i.rows, t.cells and t or ROW(i,row))) end

function COLS.add(i,row)
  for _,cols in pairs{i.x,i.y} do
    for _,col in pairs(cols) do col:add(row.cells[col.at]) end end end

function NUM.add(i,x,n)
  if x=="?" then return end
  n = n or 1
  for _=1,n  do
    if   #i.kept < i.nums   then i.ok=false; push(i.kept,x)
    elseif R() < i.nums/i.n then i.ok=false; i.kept[R(#i.kept)]=x end end end

function SYM.add(i,x,n)
  if x=="?" then return end
  i.ok = false
  i.kept[x] = n + (i.kept[x] or 0) end

--- ## Querying
function Num.ok(i)
  if not i.ok then table.sort(i.kept) end
  i.ok = true
  return i.kelp end

function Num.mid(i) local a= i:ok(); return per(a,.5) end
function Sym.mid(i)
  local mode,most = nil,-1
  for x,n in pairs(i.kept) do if n > most then most, mode = n, x end end; return mode
end

function Num.div(i) local a= i:ok(); return (per(a,.9)-per(a..1))/2.56 end
function Sym.div(i)
  local e,log=0, function(x) return math.log(x,2) end
  for x,n in pairs(i.kept) do if n > 0 then e=e- n/i.n*log(n/i.n) end end
  return e end

--- ### Column Factory
-------------------------------------------------------------------------------
local go,no={},{}

function go.CHAT() chat{aa=1,bb=3,cc={1,2,3}}; return true end

function go.ALL()
  local fails,old = 0,{}
  for k,v in pairs(the) do old[k]=v end
  for k,v in pairs(go) do
    if k~="ALL" then
      math.randomseed(the.seed or 10019)
      if v() ~= true then print("FAIL",k); fails=fails+1 end
      for k,v in pairs(old) do the[k]=v end end end
  os.exit(fails) end


(go[arg[2]] or same)()

-- local Rows=obj("Row", function(i,row) i.rows={}; i.cols=nil; i.categories={} end)
-- function Rows.add(i,row)
--   rs.kepts = rs.cols and maps(r.kepts,row,update) or i:categorize(kap(row,init) end
)
--
-- function Rows.categorize(i,cols)
--   for _,col in pairs(cols) do if not col.ignorep then
--       push(col.txt:find"[!+-]$" and i.categories.y or i.categories.y, col) end end
--   return end
--
-- function make(f,rows)
--   local function makel(row) if rows then rows:add(row) else rows=Rows(row) end
--   if type(src)=="table" then map(rows,makel) else csv(src,makel) end
--   return rows end
```

```
                                                  __
                                                 /\_\
  ____    __   __    ___ ___                     \//\ \      ___   ___     __
 /',__\ /\ \ /\ \ \   /' __` __`\        \ \ \   /\ \/\ \  /' __`\
/\__,` `\\ \ \_\ \  /\ \/\ \/\ \ __     \_\ \_  \ \ \_\ \/\ \L\.\_
\/\____/ \/`____ \ \ \_\ \_\ \_\/\_\    /\____\ \ \____/\ \__/.\_\
 \/___/    `/___/> \ \/_/\/_/\/_/\/_/    \/____/  \/___/  \/__/\/_/
                /\___/
                \/__/
```

```
-- ## class SYM: summarize symbols

local all = require"all"
local obj,push,the = all.obj, all.push, all.the
--> SYM(at:?int, txt:?str) :SYM -> Summarize a stream of non-numerics.
local SYM = obj("SYM", function(i,at,txt)
  i.at, i.txt, i.n, i.kept =  at or 0, txt or "", 0, {} end)

--> add(i:SYM: x:sum, n:?int=1) -> Add `n` count to `i.kept[n]` .
function SYM.add(i,x,n)
  if x ~= "?" then
    i.kept[x] = (n or 1) + (i.kept[x] or 0) end end

--> clone(i:SYM) :SYM -> Return a class of the same structure.
function SYM.clone(i) return SYM(i.at, i.txt) end

--> like(i:SYN,x:any,prior:num):num -> return how much `x` might belong to `i`.
function SYM.like(i,x,prior)
   return ((i.kept[x] or 0)+the.m*prior) / (i.n+the.m) end

--> mid(i:SYM):tab -> Return a columns' `mid`ddle (central tendency).
function SYM.mid(i,p)
  local max,mode=-1,nil
  for x,n in pairs(i.kept) do if n > most then most,mode = n,x end end
  return mode end

--> div(i:SYM):tab -> Return `div`ersity of a column
-- (its tendency _not_ to be a its central tendency).
function SYM.div(i,p)
  local ent, log = 0, function(x) return math.log(x,2) end
  for x,n in pairs(i.kept) do if n > 0 then ent=ent - n/i.n*log(n/i.n) end end
  return ent end

return SYM
```

```
__                                  __                 ___
/\ \__                             /\ \__            /\_ \
\ \ ,_\     __    ____   __  __    \ \ ,_\     __    \//\ \      __  __     __
 \ \ \/   /'__`\ /',__\ /\ \/\ \    \ \ \/   /'__`\    \ \ \    /\ \/\ \  /'__`\
  \ \ \_ /\  __/ /\__, `\ \ \_\ \    \ \ \_ /\  __/     \_\ \_  \ \ \_\ \/\ \L.\_
   \ \__\\ \____\\ \____/\\/`____ \    \ \__\\ \____\   /\____\ \ \____/\ \__/.\_\
    \/__/ \/____/ \/___/  `/___/> \    \/__/\/____/   \/____/  \/___/  \/__/\/_/
                                /\___/
                                \/__/


print(1)

return 1

print(2)

return 3
```