```lua
--- -------------------------------------------------------------------------
---      ___       ______                          ___
---     /\_ \     /\  ___\                        /\_ \
---     \//\ \    \ \ \__/                        \//\ \     __  __     __
---       \ \ \    \ \___``\                        \ \ \   /\ \/\ \  /'__`\
---        \_\ \_   \/\ \L\ \         __            \_\ \_ \ \ \_\ \/\ \L\.\_
---        /\____\   \ \____/        /\_\           /\____\ \ \____/\ \__/.\_\
---        \/____/    \/___/         \/_/           \/____/  \/___/  \/__/\/_/

--                                                    ___
--                                                 ,o88888
--                                              ,o8888888'
--                        ,:o:o:oooo.        ,8o88pd8888"
--                    ,.::.::o:ooooooooo. ,oo8o8pd888'"
--                  ,.:.::o:ooooooo8o8ooo.8oopd8o8o"
--                 , ..:.::o:ooooooo8ooooo.fdo8o8"
--                ,  ..:.::o:oooooo8o888o8o,cocoo"
--               , . ..:.::o:ooooooo8oooococo"
--                . ..:.::o:ooooooo8o8occcc"o
--                   . ..:.::o:oooooococcc"o:o
--                   . ..:.::o:o:,cooooco"oo:o:
--                `   . . ..:.:cocoooo"'o:o:::'
--                .`   . ..::ccccoc"'o:o:o:::'
--               :.:.    ,c:cccc"':.:.:.:.:.'
--             ..:.:"'`::::c:"'..:.:.:.:.:.'
--           ...:.'.:.::::"'    . . . . .'
--          .. . ....:."' `   .  . . ''
--        . . . ...."'
--        .. . ."'     -hrr-
--       .

local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
local the
local help=[[

lua l5.lua [OPTIONS]
(c) 2022, Tim Menzies, BSD-2-Clause
Explore the world better; explore it for good.

OPTIONS:
  -cohen     -c cohen                    =  .35
  -far       -F how far to seek poles    = .9
  -goal      -g goal class               = recurrence-events
  -keep      -k items to keep            = 256
  -K         -K manage low class counts   = 1
  -M         -M manage low evidence counts = 2
  -minItems  -m min items in a rang e    = .5
  -p         -p euclidean coefficient    = 2
  -some      -S sample size for rows     = 512
  -wait      -w wait inference some items = 10
  -want      -W range optimization goal   = plan

OPTIONS, other:
  -dump      -d stackdump on error      = false
  -file      -f data file               = ../etc/data/breastcancer.csv
  -help      -h show help               = false
  -rnd       -r round numbers           = %5.2f
  -seed      -s random number seed      = 10019
  -todo      -t start-up action         = nothing
  -n1        -n1 #repeated trials       = 20
  -n2        -n2 samples per trial      = 100
]]
```

```lua
---    ___ ___  _ ___ _  _ ___
---     |  |__/ | |   |_/  [__
---     |  |  \ | |__ | \_ ___]

---    _ _  _  _|_|_  _
---   | | |(_|  | | |_\

-- ### Maths Tricks
local r,ish,cosine

-- **r()**:  Random number shorthand.
r=math.random

-- **ish()**: is `x` is close-ish to `y`?
-- **cosine()**: for three  ABC with sides abc,
-- where does C falls on the line running AB?
function ish(x,y,z)  return math.abs(y -x ) < z end
function cosine(a,b,c)
  return math.max(0,math.min(1, (a^2+c^2-b^2)/(2*c+1E-32))) end

---    |. __|_ _
---    ||_\ | _\

-- ### List Tricks
local any,many,last,per,pop,push,sort,firsts,stsrif,copy,map,sum
local inc,inc2,inc3, has,has2,has3, powerset, shuffle
-- **any()**: returns any thing from a list
-- **many()**: return multiple **any()** things.
function any(a)        return a[ math.random(#a) ] end
function many(a,n,  u) u={}; for j=1,n do u[1+#u] =any(a) end; return u end

-- **last()**: last item in a list
-- ##per(): p-th item in a list
function last(a)       return a[ #a ] end
function per(a,p)      return a[ (p*#a)//1 ] end

-- **pop()**: dump from end
-- **push()**: add to ed
function pop(a)        return table.remove(a) end
function push(t,x)     t[1 + #t] = x; return x end

-- **sort()**: return a list, ordered on function `f`.
-- **recurrs()**:  order on sub-list first items
function sort(t,f)     table.sort(t,f); return t end
function firsts(a,b)   return a[1] < b[1] end
function stsrif(a,b)   return a[1] > b[1] end

-- **copy()**: deep copy
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

-- **map()**: return a list with `f` run over all items
function map(t,f, u) u={};for k,v in pairs(t) do u[1+#u]=f(v) end;return u end

-- **sum()**: sum all list items, filtered through `f`
-- (which defaults to just use the ran values).
function sum(t,f, n)
  n=0; map(t,function(v) n=n+(f and f(v) or v) end)
  return n end

-- **inc()** increment a 1,2, or 3 nested dictionary counter
function inc(f,a,n)      f=f or{};f[a]=(f[a] or 0) + (n or 1);  return f end
function inc2(f,a,b,n)   f=f or{};f[a]=inc( f[a] or {},b,n);  return f end
function inc3(f,a,b,c,n) f=f or{};f[a]=inc2(f[a] or {},b,c,n);return f end

-- **has()** implements a 1,2, or level nested lookup
function has(f,a)      return f[a]                      or 0 end
function has2(f,a,b)   return f[a] and has( f[a],b)   or 0 end
function has3(f,a,b,c) return f[a] and has2(f[a],b,c) or 0 end

-- **shuffle()**: randomize order (sorts in  place)
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

-- **pwoerset()**: return all subsets
function powerset(s)
  local t = {{}}
  for i = 1, #s do
    for j = 1, #t do
      t[#t+1] = {s[i],table.unpack(t[j])} end end
  return t end

---     __|_ _ . _  _  _   '~)  _|_|_ . _  _  _
---    _\ | | || |(_|_\   /_    | | ||| |(_|_\
---                 _|                     _|

-- ### String -> Things
local words, things, thing, lines

-- **words()**: split  string into list of substrings
function words(s,sep,   t)
  sep="([^" .. (sep or ",")  .. "]+)"
  t={}; for y in s:gmatch(sep) do t[1+#t] = y end; return t end

-- **things()**: convert strings in a list to things
-- **thing()**: convert string to a thing
function things(s) return map(words(s), thing) end
function thing(x)
  x = x:match"^%s*(.-)%s*$"
  if x=="true" then return true elseif x=="false" then return false end
  return tonumber(x) or x end

-- **lines()**: (iterator) return lines in a file. Standard usage is
-- `for cells in file(NAME,things) do ... end`
function lines(file,f,      x)
  file = io.input(file)
  f    = f or things
  return function() x=io.read(); if x then return f(x) else io.close(file) end e
nd end

---    _|_|_ . _  _  _   '~)  __|_ _ . _  _  _
---     | | ||| |(_|_\   /_  _\ | | || |(_|_\
---                _|                      _|

-- ### Things -> Strings
local fmt,o,oo,slots,rnds,rnd

-- **fmt()**:  String format shorthand
fmt = string.format

-- **oo()**: Print string from nested table.
-- **o()**: Generate string from nested table.
function oo(t) print(o(t)) end
function o(t,  seen, u)
  if type(t)~="table" then return tostring(t) end
  seen = seen or {}
  if seen[t] then return "..." end
  seen[t] = t
  local function show1(x) return o(x, seen) end
  local function show2(k) return fmt(":%s %s",k, o(t[k],seen)) end
  u = #t>0 and map(t,show1) or map(slots(t),show2)
  return (t.s or "").."{"..table.concat(u," ").."}" end

-- **slots()**: return table slots, sorted.
```

```
function slots(t, u)
  local 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

-- **rnds()**: round list of numbers
-- **rnd()**: round one number.
function rnds(t,f) return map(t, function(x) return nd(x,f) end) end
function rnd(x,f)
  f = not f and "%s" or number and fmt("%%%sf",f) or f
  return fmt(type(x)=="number" and (x~=x//1 and f) or "%s",x) end

---     _  _ _|__|_ . _  _  _
---    _\(/_ |  |  || |(_|_\
---                     _|

-- ### Make settings from help string  and CLI (command-line interface)
local cli

-- **cli()**: In a string, look for lines indented with two spaces, starting with a dash.
-- Each such  line should have  a long and short flag, some help tesx
-- and (at end of line), a  default values. e.g.
--
--     -seed -S set the random number seed  = 10019
--
-- Each line generates  a setting  with key "seed" and
-- default value "10019". If the command line contains one of the flags
-- (`-seed` or `-s`) then update those defaults.
function cli(help)
  local d,used = {},{}
  help:gsub("\n ([-]([^%s]+))[%s]+(-[^%s]+)[^\n]*%s([^%s]+)",
    function(long,key,short,x)
      assert(not used[short], "repeated short flag ["..short.."]")
      used[short]=short
      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] end end
       d[key] = x==true and true or thing(x) end)
  if d.help then os.exit(print(help)) end
  return d end

---    _|_ _  __|_ _
---     | (/__\ | _\

-- ### Test suites
local ok,go

-- **ok()**: maybe, print stack dump on errors.
-- Increment the `fails` counter on failed `test`.
function ok(tests,test,msg)
  print(test and "   PASS: "or "   FAIL: ",msg or "")
  if not test then
    tests.ails = tests.ails+1
    if the and the.dump then assert(test,msg) end end end

-- **go()**:  run some `tests`, controlled by `settings`.
-- Maybe update the `ails` counter.
-- Return the total fails to the operating system.
function go(settings,tests,b4,      defaults)
  tests.ails = 0
  defaults={}; for k,v in pairs(settings) do defaults[k]=v end
  local todo =  settings.todo or "all"
  for k,one in pairs(todo=="all" and slots(tests) or {todo}) do
    if k ~= "main" and type(tests[one]) == "function" then
      for k,v in pairs(defaults) do settings[k]=v end
      math.randomseed(settings.seed  or 1)
      print(fmt("#%s",one))
      tests[one](tests) end end
  if b4 then
    for k,v in pairs(_ENV) do
      if not b4[k] then print("??",k,type(v)) end end end
  os.exit(tests.ails) end

---     _ |_  . _  __|_ _
---    (_)|_) |(/_(_ | _\
---          L|

-- ### Objects
local as, is

-- **new()**:  make a new instance.
-- **class()**: define a new class of instances
as = setmetatable
function is(s,   t)
  t={tostring=o,s=s or ""}; t.index=t
  return as(t, {call=function(...) return t.new(...) end}) end


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

---    _  _  ___     ____ _  _  ____
---   |\ | |__]     |   | |\ | |___
---   | \| |__]     |___| | \| |___

local nb1, train1,test1,classify1,score1

function nb1(file)
  local i = {h={}, nh=0,e={}, names=nil, n=0, wait=the.wait, log={}}
  for row in lines(file) do
    if not i.names then i.names=row else test1(i,row); train1(i,row) end end
  return score1(i.log) end

function train1(i,t)
  i.n = i.n + 1
  if not i.h[t[#t]] then i.nh = i.nh + 1 end
  inc(i.h, t[#t])
  for col,x in pairs(t) do if x~="?" then inc3(i.e,col,x,t[#t]) end end end

function test1(i,t)
  if i.n > i.wait then push(i.log,{want=t[#t], got=classify1(i,t)}) end end

function classify1(i,t)
  local hi,out = -1
  for h,_ in pairs(i.h) do
    local prior = ((i.h[h] or 0) + the.K)/(i.n + the.K*i.nh)
    local l = prior
    for col,x in pairs(t) do
      if x ~= "?" and col ~= #t then
        l=l*(has3(i.e,col,x,h) + the.M*prior)/((i.h[h] or 0) + the.M) end end
    if l>hi then hi,out=l,h end end
  return out end

function score1(log,   n)
  n=0; for _,x in pairs(log) do if x.want==x.got then n=n+1 end end
  return n/#log end
```

```lua
--- ------------------------------------------------------------------------
---    ____ ____ ____
---    |___ | __ [__
---    |___ |__] ___]

-- ## Egs
-- Egs store examples (in `rows`), summarized in columns (in `cols`)
function Egs:new(names) return as({rows={}, cols=Cols(names)}, Egs) end

function Egs:new4file(file,  i)
  for _,row in lines(file) do if i then i:add(row) else i=Egs(row) end end
  return i end

function Egs.add(i,t)
  t = t.cells or t -- detail (for future extension)
  push(i.rows, map(i.cols.all, function(col) return col:add(t[col.at]) end)) end

function Egs.mid(i,cols) return map(cols or i.cols.all, function(col) return col:mid() end) end

function Egs.clone(i) return Egs(i.cols.names) end

function Egs.klass(i,row) return row[i.cols.klass.at] end

-- ## Col
-- Convert  names into various Column types.
local ako={}
ako.ratio  = function(x) return x:find"^[A-Z]" end
ako.goal   = function(x) return x:find"[-+!]"  end
ako.klass  = function(x) return x:find"!$"     end
ako.ignore = function(x) return x:find":$"     end
ako.less   = function(x) return x:find"-$"     end

-- Every new column goes into `all`.  Also, for any column that we we
-- are not ignoring, then that also gets added to (a) either the list
-- of `x` independent columns or `y` dependent columns; and (b) maybe,
-- the `klass` slot.
function Cols:new(names)
  local i = as({names=names, klass=nil,all={}, x={}, y={}}, Cols)
  for at,name in pairs(names) do
    local col = (ako.ratio(name) and Ratio or Nominal)(at,name)
    col.is_goal = ako.goal(name)
    push(i.all, col)
    if not ako.ignore(name) then
      if ako.klass(name) then i.klass = col end
      push(ako.goal(name) and i.y or i.x, col) end end
  return i end

-- ## Nominal
-- Summarize symbols in `Nominal`s
function Nominal:new(at,name)
  at,name = at or 0, name or ""
  return as({at=at, name=name, n=0, has={}, mode=nil, most=0}, Nominal) end

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

function Nominal.mid(i) return i.mode end

-- ## Ratio
-- Summarize numbers in `Ratio`s
function Ratio:new(at,name)
  at,name = at or 0, name or ""
  return as({at=at, name=name, n=0, mu=0, m2=0, sd=0, w=ako.less(name) and -1 or 1}, Ratio) end

function Ratio.add(i,x)
  if x ~= "?" then
    i.n =i.n+1
    local d= x - i.mu
    i.mu = i.mu + d/i.n
    i.m2 = i.m2 + d*(x - i.mu)
    i.sd = ((i.m2<0 or i.n<2) and 0) or ((i.m2/(i.n - 1))^0.5)
    i.lo = i.lo and math.min(x, i.lo) or x
    i.hi = i.hi and math.max(x, i.hi) or x end
  return x end

function Ratio.mid(i) return i.mu end

-- ## Return
return {Egs=Egs, Ratio=Ratio, Nominal=Nominal}

--- ------------------------------------------------------------------------
```

```lua
-- ## Demos
local eg={}
function eg.last(tst)
  ok(tst, 30 == last{10,20,30}, "lasts") end

function eg.per(tst,  t)
  t={};for i=1,100 do push(t,i*1000) end
  ok(tst,70000 == per(t,.7), "per") end

function eg.many(tst,  t)
  t={};for i=1,100 do push(t,i) end; many(t,10) end

function eg.sum(tst,   t)
  t={};for i=1,100 do push(t,i) end; ok(tst,5050==sum(t),"sum")end

function eg.shuffle(tst, t, good)
  t={1,2,3,4,5,6,7,8,9}
  good = true
  for j=1,10^5 do
    t= shuffle(t);
    good = good and sum(t)==45,"shuffle "..j end
  ok(tst,good, "shuffling") end

function eg.powersets(tst, t)
  ok(tst,1024==#powerset{1,2,3,4,5,6,7,8,9,10}) end

function eg.inc(tst,   f)
  f=inc3({},"a","b","c"); oo(f)
  f=inc2({},"a","b"); oo(f)
  f=inc({},"a"); oo(f)
end

function eg.nb(tst,  abcd)
  print(nb1("../etc/data/breastcancer.csv")) end


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

-- ## Stattup
the=cli(help)

go(the, eg, b4)
```