```lua
--
--                   ___
--                  /\_ \
--    __  _   _____ \//\ \     ___   _ __
--   /\ \/'\ /\ '__`\ \ \ \   / __`\/\`'__\
--   \/>  </ \ \ \L\ \ \_\ \_/\ \L\ \ \ \/
--    /\_/\_\ \ \ ,__/ /\____\ \____/\ \_\
--    \//\/_/  \ \ \/  \/____/\/___/  \/_/
--              \ \_\
--               \/_/

local b4={}; for k,v in pairs(_ENV) do b4[k]=v end;
local help=[[
XPLOR: Bayesian active learning
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua XPLOR.lua [OPTIONS]

OPTIONS:
 -f  --file  file with csv data                    = ../data/auto93.csv
 -g  --go    start-up example                      = nothing
 -h  --help  show help                             = false
 -k  --k     Bayes hack: low attribute frequency   = 2
 -m  --m     Bayes hack: low class frequency       = 1
 -s  --seed  random number seed                    = 10019]]

--[[
 | What                        | Notes                                       |
 :-----------------------------|---------------------------------------------|
 | DATA(src)                   | Store ROWs, summarize in `self.cols`        |
 | DATA:add(row)               | new row is a header, or a data row          |
 | DATA:clone(src)             | compy structure                             |
 | DATA:klass(row)             | return `row`'s class symbol.                |
 | DATA:like(row,nh,nrows)     | how much DATA likes `row`?                  |
 | DATA:stats(  nDec,cols,sDo) | get `sDo` of `cols` (round to 'nDec`)       |
 | NB:add(row)                 | update the `datas` about `row`'s klass      |
 | NB:classify(row)            | which klass likes `row` the most?           |
 | NUM(nPos,sName)             | Summarize stream of numbers                 |
 | NUM:add(x)                  | Update                                      |
 | NUM:div()                   | spread                                      |
 | NUM:like(x,...)             | how much does NUM like `x`?                 |
 | NUM:mid()                   | central tendency                            |
 | ROW(t)                      | Hold one record                             |
 | SYM(nPos,sName)             | Summarize stream of symbols.                |
 | SYM:add(s)                  | Update.                                     |
 | SYM:div()                   | spread                                      |
 | SYM:like(s,nPrior)          | how much does SYM like `n`?                 |
 | SYM:mid()                   | central tendancy                            |

CONVENTIONS: (1) The help string at top of file is parsed to create
the settings.  (2) Also, all the `go.x` functions can be run with
`lua xplor.lua -g x`.  (3) Lastly, this code's function arguments
have some type hints:

 | What                 | Notes                                          |
 :--------------------:|------------------------------------------------|
 | 2 blanks             | 2 blanks denote optional arguments             |
 | 4 blanks             | 4 blanks denote local arguments                |
 | n                    | prefix for numerics                            |
 | s                    | prefix for strings                             |
 | is                   | prefix for booleans                            |
 | fun                  | prefix for functions                           |
 | suffix s             | list of thing (so names is list of strings)    |
 | function SYM:new()   | constructor for class e.g. SYM                 |
 | e.g. sym             | denotes an instance of class constructor       |
--]]

local adds,cdf,cli,coerce,copy,csv,fmt,map
local o,obj,oo,pdf,push,rnd,run,settings,the
function obj(s,    isa,new,t)
  isa=setmetatable
  function new(k,...) local i=isa({},k); return isa(t.new(i,...) or i,k) end
  t={__tostring = function(x) return s..o(x) end}
  t.__index = t;return isa(t,{__call=new}) end

local DATA,NB,NUM,ROW,SYM=obj"DATA",obj"NUM",obj"ROW",obj"SYM",obj"NB"

function ROW:new(t) --- Hold one record
  return {cells =t} end

function SYM:new(nPos,sName) --- Summarize stream of symbols.
    return {at=nPos or 0,
            txt=sName or "",
            n=0, mode=nil,most=-1,
            has={}} end

function NUM:new(nPos,sName) --- Summarize stream of numbers
  nPos, sName=nPos or 0, sName or ""
  return {at=nPos or 0,txt=sName,n=0,
          lo= 1E32,  hi= -1E32, mu=0, m2=0, sd=0,
          w=sName:find"-$" and -1 or 1} end

function DATA:new(src) --- Store ROWs, summarize in `self.cols`
  self.rows, self.cols = {}, {names={},all={},x={},y={}}
  adds(self,src) end

function NB:new(src,reportFun)
  self.all, self.nh, self.datas = nil, 0, {}
  self.report = reportFun or function(got,want) print(got,want) end
  adds(self, src) end

-- ._ _   _   -+-  |_   _   _|  __
-- [ | )  (/,   |   [ )  (_)  (_]  _)

-- ## NUM     ----- ----- ------------------------------------------------
function NUM:add(x) --- Update
  if x ~= "?" then
    self.n  = self.n + 1
    local d = x - self.mu
    self.mu = self.mu + d/self.n
    self.m2 = self.m2 + d*(x - self.mu)
    self.sd = self.n<0 and 0 or (self.m2<0 and 0 or (self.m2/(self.n-1))^.5)
    if x > self.hi then self.hi = x end
    if x < self.lo then self.lo = x end end end

function NUM:like(x,...) --- how much does NUM like `x`?
  return self.sd>0 and pdf(x,self.mu,self.sd) or (x==self.mu and 1 or 1/big) end

function NUM:mid()  --- central tendency
  return self.mu end
function NUM:div()  --- spread
  return self.sd end

-- ## SYM     ----- ----- ------------------------------------------------
function SYM:add(s) --- Update.
  if s~="?" then
    self.n  = self.n + 1
    self.has[s] = 1 + (self.has[s] or 0)
    if self.has[s] > self.most then
      self.most,self.mode = self.has[s], s end end end

function SYM:like(s,nPrior) --- how much does SYM like `n`?
  return ((self.has[s] or 0)+the.m*nPrior) / (self.n+the.m) end

function SYM:mid()  --- central tendancy
  return self.mode end

function SYM:div(     n) --- spread
  local function fun(p) return -p*math.log(p,2) end
  e=0; for x,n in pairs(self.has) do if n>0 then e = e - fun(n/self.n) end end
  return e end

-- ## DATA     ----- ----- ------------------------------------------------
-- ### Create
function DATA:clone(src) --- compy structure
  return adds(DATA({self.cols.names}),src) end

-- ### Update
function DATA:add(row) --- new row is a header, or a data row
  if #self.cols.all==0 then self:_head(row) else self:_body(row) end end

function DATA:_head(row) --- Create `NUM`s and `SYM`s for the column headers
  self.cols.names = row
  for n,s in pairs(row) do
    local col = push(self.cols.all, (s:find"^[A-Z]" and NUM or SYM)(n,s))
    if not s:find":$" then
      if s:find"!$" then self.cols.klass= col end
      push(s:find"[!+-]" and self.cols.y or self.cols.x, col) end end end

function DATA:_body(row) --- Crete new row. Store in `rows`. Update cols.
  row = row.cells and row or ROW(row) -- Ensure `row` is a `ROW`.
  push(self.rows, row)
  for _,cols in pairs{self.cols.x, self.cols.y} do
    for _,col in pairs(cols) do
      col:add(row.cells[col.at]) end end end

function DATA:like(row,nh,nrows) --- how much DATA likes `row`?
  local prior,like,inc,x
  prior = (#self.rows + the.k) / (nrows + the.k * nh)
  like  = math.log(prior)
  row = row.cells and row.cells or row
  for _,col in pairs(self.cols.x) do
    x = row[col.at]
    if x ~= nil and x ~= "?" then
      inc  = col:like(x,prior)
      print(inc)
      like = like + math.log(inc) end end
  return like end

function DATA:klass(row) --- return `row`'s class symbol.
  return (row.cells or row.cells or row)[self.cols.klass.at] end

function DATA:stats(  nDec,cols,sDo) --- get `sDo` of `cols` (round to 'nDec`)
    local t,v
    cols, sDo = cols or self.cols.y, sDo or "mid"
    t={}; for _,col in pairs(cols) do
            v=getmetatable(col)[sDo](col)
            v=type(v)=="number" and rnd(v,nDec) or v
            t[col.txt]=v end; return t end

-- ## NB     ----- ----- ------------------------------------------------
-- ### Update
function NB:add(row) --- update the `datas` about `row`'s klass
  local function new() self.nh = self.nh+1; return self.all:clone() end
  if   self.all
  then self.all:add(row)
       local k = self.all:klass(row)
       if #self.all.rows > 10 then self.report(self:classify(row),k) end
       self.datas[k] = self.datas[k] or new()
       self.datas[k]:add(row)
  else self.all=DATA{row} end end

function NB:classify(row) --- which klass likes `row` the most?
  local most,klass,like = -math.huge
  for k,data in pairs(self.datas) do
    like = data:like(row, self.nh, #self.all.rows)
    if like > most then most,klass=like,k end end
  return klass end

-- .    .
-- |  * |_
-- |  | [_)
--[[
 | What                        | Notes                                       |
 :-----------------------------|---------------------------------------------|
 | adds(data,src)              | add list `src` or filename `src` to `data`  |
 | cdf (x)                     | Gaussian cumulative distribution            |
 | coerce(s)                   | Parse `the` config settings from `help`.    |
 | copy(t,  isShallow, u)      | copy `t` (recursive if If not `isShallaw`)  |
 | csv(sFilename, fun)         | call `fun` cells in each CSV line           |
 | fmt(str,...)                | emulate printf                              |
 | map(t1,fun)                 | apply `fun` across `t1` (skip nil results)  |
 | o(t,   seen,show,u)         |  coerce to string (skip loops, sort slots)  |
 | oo(t)                       | print nested lists                          |
 | push(t,x)                   | Push `x` to end of `t`, return `x`          |
 | rnd(n, nPlaces)             | round `n` to `nPlaces`.                     |
 | run(settings,funs)          | run one `funs`, controlled by `settings`    |
 | settings(s)                 | create a `the` variable                     |
--]]
-- ## Math     ----- ----- ------------------------------------------------
function rnd(n, nPlaces)  --- round `n` to `nPlaces`.
  local mult = 10^(nPlaces or 2)
  return math.floor(n * mult + 0.5) / mult end

function pdf(x,mu,sd) -- Gaussian probability distribution
  return math.exp(-.5*((x - mu)/sd)^2) / (sd*((2*math.pi)^0.5)) end

function cdf (x,    _cdf)  --- Gaussian cumulative distribution
  function _cdf(x,    p,t) -- Abramowitz and Stegun cdf approximation
    p = pdf(x,0,1)         -- Handbook Mathematical Functions, 1988
    t = 1 / (1+0.2316419*x)
    return (1 - p*(0.319381530*t - 0.356563782*t^2 + 1.781477937*t^3
                 - 1.821255978*t^4 + 1.330274429*t^5)) end
  return (x==0 and .5) or (x>0 and _cdf(x)) or 1-_cdf(-x) end

-- ## Lists     ----- ----- ------------------------------------------------
function push(t,x)  --- Push `x` to end of `t`, return `x`
  t[1+#t]=x; return x end

function map(t1,fun)  --- apply `fun` across `t1` (skip nil results)
  local t2={}; for _,v in pairs(t1) do t2[1+#t2] = fun(v) end; return t2 end

function copy(t,  isShallow, u) --- copy `t` (recursive if If not `isShallaw`)
  if type(t) ~= "table" then return t end
  u={}; for k,v in pairs(t) do u[k] = isShallow and v or copy(v,isShallow) end
  return setmetatable(u,getmetatable(t))  end

-- ## Strings to Things    ----- ----- ----------------------------------------
function coerce(s,    fun) --- Parse `the` config settings from `help`.
  function fun(s1)
    if s1=="true"  then return true end
    if s1=="false" then return false end
    return s1 end
  return math.tointeger(s) or tonumber(s) or fun(s:match"^%s*(.-)%s*$") end

function csv(sFilename, fun,      src,s,t) --- call `fun` cells in each CSV line
  src  = io.input(sFilename)
  while true do
    s = io.read()
    if   s
    then t = {}; for s1 in s:gmatch("([^,]+)") do t[1+#t] = coerce(s1) end
         fun(t)
    else return io.close(src) end end end

function adds(data,src)  --- add list `src` or filename `src` to `data`
  if   type(src)=="string"
  then csv(src,        function(row) data:add(row) end)
  else map(src or {}, function(row) data:add(row) end) end
  return data end

-- ## Thing to string   ----- ----- ----------------------------------------
function fmt(str,...) --- emulate printf
  return string.format(str,...) end

function oo(t) --- print nested lists
   print(o(t)) return t end

function o(t,   seen,show,u) ---  coerce to string (skip loops, sort slots)
  if type(t) ~=  "table" then return tostring(t) end
  seen=seen or {}
  if seen[t] then return "..." end
  seen[t] = t
  function show(k,v)
    if not tostring(k):find"^_"  then
      v = o(v,seen)
      return #t==0 and fmt(":%s %s",k,v) or o(v,seen) end end
  u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
  if #t==0 then table.sort(u) end
  return "{"..table.concat(u," ").."}" end

-- ## Settings     ----- ----- ------------------------------------------------
function settings(s) --- create a `the` variable
  local t,pat = {_help=s}, "\n [-][%S]+[%s]+[-][-]([%S]+)[^\n]+= ([%S]+)"
  s:gsub(pat, function(k,x) t[k]=coerce(x) end)
  return t end

function cli(t) -- Updates from command-line. Bool need no values (just flip)
  for slot,v in pairs(t) do
    v = tostring(v)
    for n,x in ipairs(arg) do
      if x=="-"..(slot:sub(1,1)) or x=="--"..slot then
        v = v=="false" and "true" or v=="true" and "false" or arg[n+1] end end
    t[slot] = coerce(v) end
  if t.help then os.exit(print("\n"..t._help.."\n")) end
  return t end

-- ## Start up     ----- ----- ------------------------------------------------
function run(settings,funs) --- run one `funs`, controlled by `settings`
  local fails,old = 0,copy(settings)
  for k,fun in pairs(funs) do
    if settings.go == "all" or settings.go == k then
      for k,v in pairs(old) do settings[k]=v end
      math.randomseed(settings.seed or 10019)
      print("# >>>>>",k)
      if fun()==false then fails = fails+1;print("# FAIL!!!!!",k); end end end
  for k,v in pairs(_ENV)do if not b4[k] then print("# rogue?",k,type(v)) end end
  os.exit(fails) end

--   .
--  _|  _  ._ _   _   __
-- (_]  (/,  [ | )  (_)  _)

local go={}
function go.the() oo(the); return  1 end

function go.cdf()
  for x=-2,2.5,.3 do print(rnd(cdf(x),4), ("-"):rep(cdf(x)*50//1)..".") end end

function go.sym(  sym)
  sym = SYM()
  for _,x in pairs{"a","a","a","a","b","b","c"} do sym:add(x) end
  return sym.mode =="a" and sym.most==4 end

function go.num(  num)
  num = NUM()
  for x=1,100 do num:add(x) end
  return 51==rnd(num.mu,0) and 29== rnd(num.sd,0) end

function go.csv()
  csv(the.file, oo); return 1 end

function go.data(     data)
  data=DATA(the.file)
  map(data.cols.x,oo); print""
  map(data.cols.y,oo) end

function go.data(     data)
  data = DATA(the.file)
  print("mid",o(data:stats(2, data.cols.x,"mid")));
  print("div",o(data:stats(2, data.cols.x,"div"))) end

function go.clone(     data1,data2)
  data1 = DATA(the.file)
  data2 = data1:clone(data1.rows)
  print("mid", o(data1:stats(2, data1.cols.x,"mid")));
  print("mid", o(data2:stats(2, data2.cols.x,"mid"))) end

local function _classify(f,nb)
  local all,correct = 0,0
  nb=NB(f,function(got,want)
          all=all+1; correct = correct + (got==want and 1 or 0) end)
  print(correct/all) end

function go.diabetes() _classify("../data/diabetes.csv") end
function go.soybean() _classify("../data/soybean.csv") end


-- ## Start  ----- ----- --------------------------------------------------------
the = settings(help)
if    pcall(debug.getlocal,4,1)
then return {the=the, NUM=NUM, SYM=SYM, DATA=DATA, ROW=ROW, NB=NB}
else   the=cli(the)
       run(the,go) end
```