```lua
local b4={}; for k,v in pairs(_ENV) do b4[k]=v end -- LUA trivia. Ignore.
local help=[[
CSV : summarized csv file
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua seen.lua [OPTIONS]

OPTIONS:
 -e  --eg        start-up example                     = nothing
 -d  --dump      on test failure, exit with stack dump = false
 -f  --file      file with csv data                   = ../data/auto93.csv
 -h  --help      show help                            = false
 -k  --k         bayes low frequency factor           = 2
 -m  --m         bayes low frequency factor           = 1
 -n  --nums      number of nums to keep               = 512
 -p  --p         distance calculation coefficient     = 2
 -s  --seed      random number seed                   = 10019
 -S  --seperator feild seperator                      = , ]]
```

Function argument conventions:

1. two blanks denote optionas, four blanls denote locals:
2. prefix n,s,is,fun denotes number,string,bool,function;
3. suffix s means list of thing (so names is list of strings)
4. c is a column index (usually)

Misc routines

Handle Settings

```lua
local the,coerce,cli
```

Parse the config settings from help.

```lua
function coerce(s,    fun)
  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
```

Create a the variables

```lua
the={}
help:gsub("\n [-][%S]+[%s]+[-][-]([%S]+)[^\n]+= ([%S]+)",
          function(k,x) the[k]=coerce(x) end)
```

Update settings from values on command-line flags. Booleans need no values (we just flip the defeaults).

```lua
function cli(t)
  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"..help.."\n")) end
  return t end
```

Lists

```lua
local copy,per,push,csv,sort,map,lt
```

Deepcopy

```lua
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 setmetatable(u,getmetatable(t))  end
```

Return the p-th thing from the sorted list t.

```lua
function per(t,p)
  p=math.floor(((p or .5)*#t)+.5); return t[math.max(1,math.min(#t,p))] end
```

Add to t, return x.

```lua
function push(t,x) t[1+#t]=x; return x end
```

Function, return a sorted list.

```lua
function sort(t,f) table.sort(t,f); return t end
```

Sorting function

```lua
function lt(x) return function(t1,t2) return t1[x] < t2[x] end end
```

Map a function over a list

```lua
function map(t1,fun,    t2)
  t2={}; for _,v in pairs(t1) do t2[1+#t2] = fun(v) end; return t2 end
```

Call fun on each row. Row cells are divided in the.seperator.

```lua
function csv(fname,fun,      sep,src,s,t)
  sep = "([^" .. the.seperator .. "]+)"
  src = io.input(fname)
  while true do
    s = io.read()
    if not s then return io.close(src) else
      t={}
      for s1 in s:gmatch(sep) do t[1+#t] = coerce(s1) end
      fun(t) end end end
```

Strings

```lua
local o,oo
```

o is a telescopt and oo are some binoculars we use to exam stucts. o: generates a string from a nested table.

```lua
function o(t,   show,u)
  if type(t) ~=  "table" then return tostring(t) end
  function show(k,v)
    if not tostring(k):find"^_"  then
      v = o(v)
      return #t==0 and string.format(":%s %s",k,v) or tostring(v) 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
```

oo: prints the string from o.

```lua
function oo(t) print(o(t)) return t end
```

Misc

```lua
local rogues, rnd,  obj
```

Find rogue locals.

```lua
function rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end
```

Maths

```lua
function rnd(x, places)
  local mult = 10^(places or 2)
  return math.floor(x * mult + 0.5) / mult end
```

obj(“Thing”) enables a constructor Thing:new() … and a pretty-printer for Things.

```lua
function obj(s,    t,i,new)
  function new(k,...) i=setmetatable({},k);
                      return setmetatable(t.new(i,...) or i,k) end
  t={__tostring = function(x) return s..o(x) end}
  t.__index = t;return setmetatable(t,{__call=new}) end
```

Objects

```lua
local Cols,Data,Num,Row,Sym=obj"Cols",obj"Data",obj"Num",obj"Rows",obj"Sym"
```

Syms summarize a stream of symbols.

```lua
function Sym:new(c,s)
  return {n=0,          -- items seen
          at=c or 0,    -- column position
          name=s or "", -- column name
          _has={}       -- kept data
         } end
```

Num ummarizes a stream of numbers.

```lua
function Num:new(c,s)
  return {n=0,at=c or 0, name=s or "", _has={}, -- as per Sym
          lo= math.huge,   -- lowest seen
          hi= -math.huge,  -- highest seen
          isSorted=true,   -- no updates since last sort of data
          w = ((s or ""):find"-$" and -1 or 1)
         } end
```

Columns Holds of summaries of columns. Columns are created once, then may appear in multiple slots.

```lua
function Cols:new(names)
  self.names=names -- all column names
  self.all={}      -- all the columns (including the skipped ones)
  self.klass=nil   -- the single dependent klass column (if it exists)
  self.x={}        -- independent columns (that are not skipped)
  self.y={}        -- depedent columns (that are not skipped)
  for c,s in pairs(names) do
    local col = push(self.all, -- Numerics start with Uppercase.
                    (s:find"^[A-Z]*" and Num or Sym)(c,s))
    if not s:find":$" then -- some columns are skipped
       push(s:find"[!+-]" and self.y or self.x, col) -- some cols are goal cols
       if s:find"!$" then self.klass=col end end end end
```

Row holds one record

```lua
function Row:new(t) return {cells=t,          -- one record
                            cooked=copy(t), -- used if we discretize data
                            isEvaled=false    -- true if y-values evaluated.
                           } end
```

Data is a holder of rows and their sumamries (in cols).

```lua
function Data:new(src)
  self.cols = nil -- summaries of data
  self.rows = {}  -- kept data
  if   type(src) == "string"
  then csv(src, function(row) self:add(row) end)
  else for _,row in pairs(src or {}) do self:add(row) end end end
```

Sym

Add one thing to col. For Num, keep at most nums items.

```lua
function Sym:add(v)
  if v~="?" then self.n=self.n+1; self._has[v]= 1+(self._has[v] or 0) end end

function Sym:mid(col,    most,mode)
  most=-1; for k,v in pairs(self._has) do if v>most then mode,most=k,v end end
  return mode end
```

distance between two values.

```lua
function Sym:dist(v1,v2)
  return  v1=="?" and v2=="?" and 1 or v1==v2 and 0 or 1 end
```

Diversity measure for symbols = entropy.

```lua
function Sym:div(    e,fun)
  function fun(p) return p*math.log(p,2) end
  e=0; for _,n in pairs(self._has) do if n>0 then e=e - fun(n/self.n) end end
  return e end

  -- Return how much `x` might belong to `self`.
function SYM:like(x,prior)
   return ((self._has[x] or 0)+the.m*prior) / (self.n+the.m) end
```

Num

Return kept numbers, sorted.

```lua
function Num:nums()
  if not self.isSorted then table.sort(self._has); self.isSorted=true end
  return self._has end
```

Reservoir sampler. Keep at most the.nums numbers (and if we run out of room, delete something old, at random).,

```lua
function Num:add(v,    pos)
  if v~="?" then
    self.n  = self.n + 1
    self.lo = math.min(v, self.lo)
    self.hi = math.max(v, self.hi)
    if     #self._has < the.nums           then pos=1 + (#self._has)
    elseif math.random() < the.nums/self.n then pos=math.random(#self._has) end
    if pos then self.isSorted = false
                self._has[pos] = tonumber(v) end end end
```

distance between two values.

```lua
function Num:dist(v1,v2)
  if   v1=="?" and v2=="?" then return 1 end
  v1,v2 = self:norm(v1), self:norm(v2)
  if v1=="?" then v1 = v2<.5 and 1 or 0 end
  if v2=="?" then v2 = v1<.5 and 1 or 0 end
  return math.abs(v1-v2) end
```

Return middle

```lua
function Num:mid() return per(self:nums(), .5) end
```

Return diversity

```lua
function Num:div() return (per(self:nums(), .9) - per(self:nums(),.1))/2.58 end
```

Normalized numbers 0..1. Everything else normalizes to itself.

```lua
function Num:norm(n)
  return x=="?" and x or (n-self.lo)/(self.hi-self.lo + 1E-32) end
```

Return the likelihood that `x` belongs to `i`. <

```lua
function NUM:like(x,...)
  local sd,mu=self:div(), self: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
```

Data

Add a `row` to `data`. Calls `add()` to updatie the `cols` with new values.

```lua
function Data:add(xs,    row)
 if   not self.cols
 then self.cols = Cols(xs)
 else row= push(self.rows, xs.cells and xs or Row(xs)) -- ensure xs is a Row
      for _,todo in pairs{self.cols.x, self.cols.y} do
        for _,col in pairs(todo) do
          col:add(row.cells[col.at]) end end end end
```

Return a new `Data` that mimics structure of `self`. Add `src` to the clone.

```lua
function Data:clone(  src,    out)
  out = Data()
  out:add(self.cols.name)
  for _,row in pairs(src or {}) do out:add(row) end
  return out end
```

For `showCols` (default=`data.cols.x`) in `data`, show `fun` (default=`mid`), rounding numbers to `places` (default=2)

```lua
function Data:stats(  places,showCols,fun,    t,v)
  showCols, fun = showCols or self.cols.y, fun or "mid"
  t={}; for _,col in pairs(showCols) do
          v=fun(col)
          v=type(v)=="number" and rnd(v,places) or v
          t[col.name]=v end; return t end
```

Distance between rows (returns 0..1). For unknown values, assume max distance.

```lua
function Data:dist(row1,row2)
  local d = 0
  for _,col in pairs(self.cols.x) do
    d = d + col:dist(row1.cells[col.at], row2.cells[col.at])^the.p end
  return (d/#self.cols.x)^(1/the.p) end
```

Sort `rows` (default=`data.rows`) by distance to `row1`.

```lua
function Data:around(row1,  rows,     fun)
  function fun(row2) return {row=row2, dist=self:dist(row1,row2)} end
  return sort(map(rows or self.rows, fun),lt"dist") end
```

Return `P(H)\*P(E1|H))\*p(E2|H)....` Work in logs (to cope with small nums)

```lua
function Data:like(row, nklasses, nrows)
  local prior,like,inc,x
  prior = (#self.rows + the.k) / (nrows + the.k * nklasses)
  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)
      like = like + math.log(inc) end end
  return like end
```

Test Engine

```lua
local eg, fails = {},0
```

1. reset random number seed before running something.
2. Cache the detaults settings, and…
3. … restore them after the test
4. Print error messages or stack dumps as required.
5. Return true if this all went well.

```lua
local function runs(k,     old,status,out,msg)
  if not eg[k] then return end
  math.randomseed(the.seed) -- reset seed [1]
  old={}; for k,v in pairs(the) do old[k]=v end --  [2]
  if the.dump then -- [4]
    status,out=true, eg[k]()
  else
    status,out=pcall(eg[k]) -- pcall means we do not crash and dump on errror
  end
  for k,v in pairs(old) do the[k]=v end -- restore old settings [3]
  msg = status and ((out==true and "PASS") or "FAIL") or "CRASH" -- [4]
  print("!!!!!!", msg, k, status)
  return out or err end
```

Tests

Test that the test happes when something crashes?

```lua
function eg.BAD() print(eg.dont.have.this.field) end
```

Sort all test names.

```lua
function eg.LIST(   t)
  t={}; for k,_ in pairs(eg) do t[1+#t]=k end; table.sort(t); return t end

  -- List test names.
function eg.LS()
  print("\nExamples lua csv -e ...")
  for _,k in pairs(eg.LIST()) do print(string.format("\t%s",k)) end
  return true end
```

Run all tests

```lua
function eg.ALL()
  for _,k in pairs(eg.LIST()) do
    if k ~= "ALL" then
      print"\n-----------------------------------"
      if not runs(k) then fails=fails+ 1 end end end
  return true end
```

Settings come from big string top of "sam.lua" (maybe updated from comamnd line)

```lua
function eg.the() oo(the); return true end
```

The middle and diversity of a set of symbols is called "mode" and "entropy" (and the latter is zero when all the symbols are the same).

```lua
function eg.sym(  sym,entropy,mode)
  sym= Sym()
  for _,x in pairs{"a","a","a","a","b","b","c"} do sym:add(x) end
  mode, entropy = sym:mid(), sym:div()
  entropy = (1000*entropy)//1/1000
  oo({mid=mode, div=entropy})
  return mode=="a" and 1.37 <= entropy and entropy <=1.38 end
```

The middle and diversity of a set of numbers is called "median" and "standard deviation" (and the latter is zero when all the nums are the same).

```lua
function eg.num(  num,mid,div)
  num=Num()
  for i=1,100 do num:add(i) end
  mid,div = num:mid(), num:div()
  print(mid ,div)
  return 50<= mid and mid<= 52 and 30.5 <div and div<32 end
```

Nums store only a sample of the numbers added to it (and that storage is done such that the kept numbers span the range of inputs).

```lua
function eg.bignum(  num)
  num=Num()
  the.nums = 32
  for i=1,1000 do num:add(i) end
  oo(num:nums())
  return 32==#num._has; end
```

Show we can read csv files.

```lua
function eg.csv(   n)
  n=0
  csv("../data/auto93.csv",function(row)
    n=n+1; if n> 10 then return else oo(row) end end); return true end
```

Can I load a csv file into a Data?.

```lua
function eg.data(   d)
  d = Data("../data/auto93.csv")
  for _,col in pairs(d.cols.y) do oo(col) end
  return true
end
```

Print some stats on columns.

```lua
function eg.stats(   data,mid,div)
  data = Data("../data/auto93.csv")
  div  = function(col) return col:div() end
  mid  = function(col) return col:mid() end
  print("xmid", o( data:stats(2, data.cols.x, mid)))
  print("xdiv", o( data:stats(3, data.cols.x, div)))
  print("ymid", o( data:stats(2, data.cols.y, mid)))
  print("ydiv", o( data:stats(3, data.cols.y, div)))
  return true
end
```

distance functions

```lua
function eg.around(    data,around)
  data = Data("../data/auto93.csv")
  around = data:around(data.rows[1] )
  for i=1,380,40 do print(around[i].dist, o(around[i].row.cells)) end
  return true end
```

Start up

```lua
the = cli(the)
runs(the.eg)
rogues()
os.exit(fails)
```

That's all folks.