```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
 -n  --nums      number of nums to keep                = 512
 -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
local the,coerce,cli
-- Parse `the` config settings from `help`.
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
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).
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

-- ### Linting code
-- ### Lists
local copy,per,push,csv
-- deepcopy
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`.
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`.
function push(t,x) t[1+#t]=x; return x end

-- ## Call `fun` on each row. Row cells are divided in `the.seperator`.
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
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.
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`.
function oo(t) print(o(t)) return t end

-- ### Misc
local rogues, rnd,  obj
--- Find rogue locals.
function rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end

-- ### Maths
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.
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
```

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

-- `Sym`s summarize a stream of symbols.
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.
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.
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
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`).
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
```

```lua
-- ---------------------------------------
-- ## Sym
-- Add one thing to `col`. For Num, keep at most `nums` items.
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

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

-- ---------------------------------------
-- ## Num
-- Return kept numbers, sorted.
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).,
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
--
-- Diversity (standard deviation for Nums, entropy for Syms)
function Num:div(    a)  a=self:nums(); return (per(a,.9)-per(a,.1))/2.58 end

-- Central tendancy (median for Nums, mode for Syms)
function Num:mid() return per(self:nums(),.5) end
```

```lua

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

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

```lua

-- ---------------------------------
-- ## Test Engine
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.
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?
function eg.BAD() print(eg.dont.have.this.field) end

-- Sort all test names.
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
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)
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).
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).
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).
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.
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?.
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.
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

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