```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                       = ,]]

-- ## Misc routines
-- ### Handle Settings
-- Parse `the` config settings from `help`.
local the={}
local function coerce(s)
  local function coerce1(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 coerce1(s:match"^%s*(.-)%s*$") end

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

-- ### Strings
-- `o` generates a string from a nested table.
local function o(t)
  if type(t) ~=  "table" then return tostring(t) end
  local 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
  local u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
  if #t==0 then table.sort(u) end
  return (t._is or "").."{"..table.concat(u," ").."}" end

-- `oo` prints the string from `o`.
local function oo(t) print(o(t)) return t end

-- ### Lists
-- Deepcopy
local function copy(t)
  if type(t) ~= "table" then return t end
  local 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`.
local 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`.
local function push(t,x) t[1+#t]=x; return x end

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

```lua
-- ----------------------------------------
-- ## Objects
-- `Data` is a holder of `rows` and their sumamries (in `cols`).
local function Data() return {_is = "Data",
                              cols= nil,  -- summaries of data
                              rows= {}    -- kept data
                             } end

-- `Columns` Holds of summaries of columns.
-- Columns are created once, then may appear in  multiple slots.
local function Cols() return {
  _is  = "Cols",
  names={},  -- all column names
  all={},    -- all the columns (including the skipped ones)
  klass=nil, -- the single dependent klass column (if it exists)
  x={},      -- independent columns (that are not skipped)
  y={}       -- depedent columns (that are not skipped)
  } end

-- `Sym`s summarize a stream of symbols.
local function Sym(c,s)
  return {_is= "Sym",
          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.
local function Num(c,s)
  return {_is="Nums",
          n=0,at=c or 0, name=s or "", _has={}, -- as per Sym
          isNum=true,      -- mark that this is a number
          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

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

-- ## Data
-- Add one thing to `col`. For Num, keep at most `nums` items.
local function add(col,v)
  if v~="?" then
    col.n = col.n + 1
    if not col.isNum then col._has[v] = 1 + (col._has[v] or 0) else
       col.lo = math.min(v, col.lo)
       col.hi = math.max(v, col.hi)
       local pos
       if     #col._has < the.nums           then pos = 1 + (#col._has)
       elseif math.random() < the.nums/col.n then pos = math.random(#col._has) end
       if pos then col.isSorted = false
                   col._has[pos] = tonumber(v) end end end end

local function adds(col,t) for _,x in pairs(t) do add(col,x) end; return col end

--- Add a `row` to `data`. Calls `add()` to  updatie the `cols` with new values.
local function record(data,xs)
  local row= push(data.rows, xs.cells and xs or Row(xs)) -- ensure xs is a Row
  for _,todo in pairs{data.cols.x, data.cols.y} do
    for _,col in pairs(todo) do
      add(col, row.cells[col.at]) end end end

--- Generate rows from some `src`.  If `src` is a string, read rows from file;
-- else read rows from a `src`  table. When reading, use row1 to define columns.
local  function records(src,      data,head,body)
  function head(sNames)
    local cols = Cols()
    cols.names = namess
    for c,s in pairs(sNames) do
      local col = push(cols.all, -- Numerics start with Uppercase.
                       (s:find"^[A-Z]*" and Num or Sym)(c,s))
      if not s:find":$" then -- some columns are skipped
        push(s:find"[!+-]" and cols.y or cols.x, col) -- some cols are goal cols
        if s:find"!$"    then cols.klass=col end end end
    return cols
  end ------------
  function body(t) -- treat first row differently (defines the columns)
    if data.cols then record(data,t) else data.cols=head(t) end
  end ----------
  data =  Data()
  if type(src)=="string" then csv(src, body) else
    for _,t in pairs(src or {}) do body(t) end end
  return data end

-- ### Query
-- Return kept numbers, sorted.
local function nums(num)
  if not num.isSorted then table.sort(num._has); num.isSorted=true end
  return num._has end

-- Diversity (standard deviation for Nums, entropy for Syms)
local function div(col)
  if  col.isNum then local a=nums(col); return (per(a,.9)-per(a,.1))/2.58 else
    local function fun(p) return p*math.log(p,2) end
    local e=0
    for _,n in pairs(col._has) do if n>0 then e=e-fun(n/col.n) end end
    return e end end

-- Central tendancy (median for Nums, mode for Syms)
local function mid(col)
  if col.isNum then return per(nums(col),.5) else
    local most,mode = -1
    for k,v in pairs(col._has) do if v>most then mode,most=k,v end end
    return mode end end

-- Diversity (standard deviation for Nums, entropy for Syms)
local function div(col)
  if  col.isNum then local a=nums(col); return (per(a,.9)-per(a,.1))/2.58 else
    local function fun(p) return p*math.log(p,2) end
    local e=0
    for _,n in pairs(col._has) do if n>0 then e=e-fun(n/col.n) end end
    return e end end

-- For `showCols` (default=`data.cols.x`) in `data`, report `fun` (default=`mid`).
local function stats(data,  showCols,fun,    t)
  showCols, fun = showCols or data.cols.y, fun or mid
  t={}; for _,col in pairs(showCols) do t[col.name]=fun(col) 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.
-- 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
    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= adds(Sym(), {"a","a","a","a","b","b","c"})
  mode, entropy = mid(sym), div(sym)
  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)
  num=Num()
  for i=1,100 do add(num,i) end
  local med,ent = mid(num), div(num)
  print(mid(num) ,div(num))
  return 50<= med and med<= 52 and 30.5 <ent and ent <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 add(num,i) end
  oo(nums(num))
  return 32==#num._has; end

-- Show we can read csv files.
function eg.csv()
  local n=0
  csv("../data/auto93.csv",function(row)
    n=n+1; if n> 10 then return else oo(row) end end); return true end

-- Print some stats on columns.
function eg.stats()
  oo(stats(records("../data/auto93.csv"))); return true end

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