```lua
1   local b4={}; for k,v in pairs(_ENV) do b4[k]=v end -- LUA trivia. Ignore.
2   local help=[[
3   CSV : summarized csv file
4   (c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license
5
6   USAGE: lua seen.lua [OPTIONS]
7
8   OPTIONS:
9    -e  --eg        start-up example              = nothing
10   -d  --dump      on test failure, exit with stack dump = false
11   -f  --file      file with csv data            = ../data/auto93.csv
12   -h  --help      show help                     = false
13   -n  --nums      number of nums to keep        = 512
14   -s  --seed      random number seed            = 10019
15   -S  --seperator feild seperator               = ,]]
16
17   -- Function argument conventions:
18   -- 1. two blanks denote optionas, four blanls denote locals:
19   -- 2. prefix n,s,is,fun denotes number,string,bool,function;
20   -- 3. suffix s means list of thing (so names is list of strings)
21   -- 4. c is a column index (usually)
```

```lua
22   -- ## Misc routines
23   -- ### Handle Settings
24   local the,coerce,cli
25   -- Parse `the` config settings from `help`.
26   function coerce(s,     fun)
27     function fun(s1)
28       if s1=="true"  then return true end
29       if s1=="false" then return false end
30       return s1 end
31     return math.tointeger(s) or tonumber(s) or fun(s:match"^%s*(.-)%s*$") end
32
33   -- Create a `the` variables
34   the={}
35   help:gsub("\n [-][%S]+[%s]+[-][-]([%S]+)[^\n]+= ([%S]+)",
36                 function(k,x) the[k]=coerce(x) end)
37
38   -- Update settings from values on command-line flags. Booleans need no values
39   -- (we just flip the defeaults).
40   function cli(t)
41     for slot,v in pairs(t) do
42       v = tostring(v)
43       for n,x in ipairs(arg) do
44         if x=="-"..(slot:sub(1,1)) or x=="--"..slot then
45           v = v=="false" and "true" or v=="true" and "false" or arg[n+1] end end
46       t[slot] = coerce(v) end
47     if t.help then os.exit(print("\n"..help.."\n")) end
48     return t end
49
50   -- ### Lists
51   local copy,per,push,csv
52   -- deepcopy
53   function copy(t,      u)
54     if type(t) ~= "table" then return t end
55     u={}; for k,v in pairs(t) do u[k] = copy(v) end
56     return setmetatable(u,getmetatable(t))   end
57
58   -- Return the `p`-th thing from the sorted list `t`.
59   function per(t,p)
60     p=math.floor(((p or .5)*#t)+.5); return t[math.max(1,math.min(#t,p))] end
61
62   -- Add to `t`, return `x`.
63   function push(t,x) t[1+#t]=x; return x end
64
65   -- ## Call `fun` on each row. Row cells are divided in `the.seperator`.
66   function csv(fname,fun,      sep,src,s,t)
67     sep = "([^" .. the.seperator .. "]+)"
68     src = io.input(fname)
69     while true do
70       s = io.read()
71       if not s then return io.close(src) else
72         t={}
73         for s1 in s:gmatch(sep) do t[1+#t] = coerce(s1) end
74         fun(t) end end end
75
76   -- ### Strings
77   local o,oo
78   -- `o` is a telescopt and `oo` are some binoculars we use to exam stucts.
79   -- `o`: generates a string from a nested table.
80   function o(t,    show,u)
81     if type(t) ~= "table" then return tostring(t) end
82     function show(k,v)
83       if not tostring(k):find"^_" then
84         v = o(v)
85         return #t==0 and string.format(":%s %s",k,v) or tostring(v) end end
86     u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
87     if #t==0 then table.sort(u) end
88     return "{"..table.concat(u," ").."}" end
89
90   -- `oo`: prints the string from `o`.
91   function oo(t) print(o(t)) return t end
92
93   -- ### Misc
94   local rogues, rnd,  obj
95   --- Find rogue locals.
96   function rogues()
97     for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end
98
99   -- ### Maths
100  function rnd(x, places)
101    local mult = 10^(places or 2)
102    return math.floor(x * mult + 0.5) / mult end
103
104  -- obj("Thing") enables a constructor Thing:new() ... and a pretty-printer
105  -- for Things.
106  function obj(s,     t,i,new)
107    function new(k,...) i=setmetatable({},k);
108                        return setmetatable(t.new(i,...) or i,k) end
109    t={__tostring = function(x) return s..o(x) end}
110    t.__index = t;return setmetatable(t,{__call=new}) end
```

```lua
111  -- ----------------------------------------
112  -- ## Objects
113  local Cols,Data,Num,Row,Sym=obj"Cols",obj"Data",obj"Num",obj"Rows",obj"Sym"
114
115  -- `Sym`s summarize a stream of symbols.
116  function Sym:new(c,s)
117    return {n=0,           -- items seen
118            at=c or 0,     -- column position
119            name=s or "",  -- column name
120            _has={}        -- kept data
121           } end
122
123  -- `Num` ummarizes a stream of numbers.
124  function Num:new(c,s)
125    return {n=0,at=c or 0, name=s or "", _has={}, -- as per Sym
126            lo= math.huge,   -- lowest seen
127            hi= -math.huge,  -- highest seen
128            isSorted=true,   -- no updates since last sort of data
129            w = ((s or ""):find"-$" and -1 or 1)
130           } end
131
132  -- `Columns` Holds of summaries of columns.
133  -- Columns are created once, then may appear in  multiple slots.
134  function Cols:new(names)
135    self.names=names -- all column names
136    self.all={}      -- all the columns (including the skipped ones)
137    self.klass=nil   -- the single dependent klass column (if it exists)
138    self.x={}        -- independent columns (that are not skipped)
139    self.y={}        -- depedent columns (that are not skipped)
140    for c,s in pairs(names) do
141      local col = push(self.all, -- Numerics start with Uppercase.
142                       (s:find"^[A-Z]"" and Num or Sym)(c,s))
143      if not s:find"$" then -- some columns are skipped
144        push(s:find"[!+-]" and self.y or self.x, col) -- some cols are goal cols
145        if s:find"!$" then self.klass=col end end end end
146
147  -- `Row` holds one record
148  function Row:new(t) return {cells=t,              -- one record
149                              cooked=copy(t), -- used if we discretize data
150                              isEvaled=false    -- true if y-values evaluated.
151                             } end
152
153  -- `Data` is a holder of `rows` and their sumamries (in `cols`).
154  function Data:new(src)
155    self.cols = nil -- summaries of data
156    self.rows = {}  -- kept data
157    if   type(src) == "string"
158    then csv(src, function(row) self:add(row) end)
159    else for _,row in pairs(src or {}) do self:add(row) end end end
```

Printed by Timothy Menzies

Aug 27, 22 19:19   **csv.lua**   Page 4/7
Aug 27, 22 19:19   **csv.lua**   Page 5/7
Aug 27, 22 19:19   **csv.lua**   Page 6/7

```lua
160 -- -------------------------------------------
161 -- ## Sym
162 -- Add one thing to `col`. For Num, keep at most `nums` items.
163 function Sym:add(v)
164   if v~="?" then self.n=self.n+1; self._has[v]= 1+(self._has[v] or 0) end end
165
166 function Sym:mid(col,    most,mode)
167   most=-1; for k,v in pairs(self._has) do if v>most then mode,most=k,v end end
168   return mode end
169
170 function Sym:div(    e,fun)
171   function fun(p) return p*math.log(p,2) end
172   e=0; for _,n in pairs(self._has) do if n>0 then e=e - fun(n/self.n) end end
173   return e end
174
175 -- -------------------------------------------
176 -- ## Num
177 -- Return kept numbers, sorted.
178 function Num:nums()
179   if not self.isSorted then table.sort(self._has); self.isSorted=true end
180   return self._has end
181
182 -- Reservoir sampler. Keep at most `the.nums` numbers
183 -- (and if we run out of room, delete something old, at random).,
184 function Num:add(v,    pos)
185   if v~="?" then
186     self.n  = self.n + 1
187     self.lo = math.min(v, self.lo)
188     self.hi = math.max(v, self.hi)
189     if     #self._has < the.nums        then pos=1 + (#self._has)
190     elseif math.random() < the.nums/self.n then pos=math.random(#self._has) end
191     if pos then self.isSorted = false
192                 self._has[pos] = tonumber(v) end end end
193 --
194 -- Diversity (standard deviation for Nums, entropy for Syms)
195 function Num:div(    a)  a=self:nums(); return (per(a,.9)-per(a,.1))/2.58 end
196
197 -- Central tendancy (median for Nums, mode for Syms)
198 function Num:mid() return per(self:nums(),.5) end
```

```lua
198
199 -- -------------------------------------------
200 -- ## Data
201 -- Add a `row` to `data`. Calls `add()` to  updatie the `cols` with new values.
202 function Data:add(xs,    row)
203   if   not self.cols
204   then self.cols = Cols(xs)
205   else row= push(self.rows, xs.cells and xs or Row(xs)) -- ensure xs is a Row
206     for _,todo in pairs{self.cols.x, self.cols.y} do
207       for _,col in pairs(todo) do
208         col:add(row.cells[col.at]) end end end end
209
210 -- For `showCols` (default=`data.cols.x`) in `data`, show `fun` (default=`mid`),
211 -- rounding numbers to `places` (default=2)
212 function Data:stats( places,showCols,fun,    t,v)
213   showCols, fun = showCols or self.cols.y, fun or "mid"
214   t={}; for _,col in pairs(showCols) do
215         v=fun(col)
216         v=type(v)=="number" and rnd(v,places) or v
217         t[col.name]=v end; return t end
```

```lua
218
219 -- -------------------------------------------
220 -- ## Test Engine
221 local eg, fails = {},0
222
223 -- 1. reset random number seed before running something.
224 -- 2. Cache the defaults settings, and...
225 -- 3. ... restore them after the test
226 -- 4. Print error messages or stack dumps as required.
227 -- 5. Return true if this all went well.
228 local function runs(k,    old,status,out,msg)
229   if not eg[k] then return end
230   math.randomseed(the.seed) -- reset seed [1]
231   old={}; for k,v in pairs(the) do old[k]=v end --  [2]
232   if the.dump then --  [4]
233     status,out=true, eg[k]()
234   else
235     status,out=pcall(eg[k]) -- pcall means we do not crash and dump on errror
236   end
237   for k,v in pairs(old) do the[k]=v end -- restore old settings [3]
238   msg = status and ((out==true and "PASS") or "FAIL") or "CRASH" --  [4]
239   print("!!!!!!", msg, k, status)
240   return out or err end
241
242 -- -------------------------------------------
243 -- ## Tests
244 -- Test that the test  happes when something crashes?
245 function eg.BAD() print(eg.dont.have.this.field) end
246
247 -- Sort all test names.
248 function eg.LIST(   t)
249   t={}; for k,_ in pairs(eg) do t[1+#t]=k end; table.sort(t); return t end
250
251 -- List test names.
252 function eg.LS()
253   print("\nExamples lua csv -e ...")
254   for _,k in pairs(eg.LIST()) do print(string.format("\t%s",k)) end
255   return true end
256
257 -- Run all tests
258 function eg.ALL()
259   for _,k in pairs(eg.LIST()) do
260     if k ~= "ALL" then
261       print"\n----------------------------------"
262       if not runs(k) then fails=fails+ 1 end end end
263   return true end
```

```lua
264  -- Settings come from big string top of "sam.lua"
265  -- (maybe updated from comamnd line)
266  function eg.the() oo(the); return true end
267
268  -- The middle and diversity of a set of symbols is called "mode"
269  -- and "entropy" (and the latter is zero when all the symbols
270  -- are the same).
271  function eg.sym(  sym,entropy,mode)
272    sym= Sym()
273    for _,x in pairs{"a","a","a","a","b","b","c"} do sym:add(x) end
274    mode, entropy = sym:mid(), sym:div()
275    entropy = (1000*entropy)//1/1000
276    oo({mid=mode, div=entropy})
277    return mode=="a" and 1.37 <= entropy and entropy <=1.38 end
278
279  -- The middle and diversity of a set of numbers is called "median"
280  -- and "standard deviation" (and the latter is zero when all the nums
281  -- are the same).
282  function eg.num(  num,mid,div)
283    num=Num()
284    for i=1,100 do num:add(i) end
285    mid,div = num:mid(), num:div()
286    print(mid ,div)
287    return 50<= mid and mid<= 52 and 30.5 <div and div<32 end
288
289  -- Nums store only a sample of the numbers added to it (and that storage
290  -- is done such that the kept numbers span the range of inputs).
291  function eg.bignum(  num)
292    num=Num()
293    the.nums = 32
294    for i=1,1000 do num:add(i) end
295    oo(num:nums())
296    return 32==#num._has; end
297
298  -- Show we can read csv files.
299  function eg.csv(  n)
300    n=0
301    csv("../data/auto93.csv",function(row)
302      n=n+1; if n> 10 then return else oo(row) end end); return true end
303
304  -- Can I load a csv file into a Data?.
305  function eg.data(  d)
306    d = Data("../data/auto93.csv")
307    for _,col in pairs(d.cols.y) do oo(col) end
308    return true
309  end
310
311  -- Print some stats on columns.
312  function eg.stats(  data,mid,div)
313    data = Data("../data/auto93.csv")
314    div=function(col) return col:div() end
315    mid=function(col) return col:mid() end
316    print("xmid", o( data:stats(2,data.cols.x, mid)))
317    print("xdiv", o( data:stats(3,data.cols.x, div)))
318    print("ymid", o( data:stats(2,data.cols.y, mid)))
319    print("ydiv", o( data:stats(3,data.cols.y, div)))
320    return true
321  end
322
323  -- --------------------------------
324  the = cli(the)
325  runs(the.eg)
326  rogues()
327  os.exit(fails)
```