

```

1  -----
2  ---
3  ---
4  ---
5  ---
6  ---
7  ---
8  ---
9  ---
10 ---
11 --- a little LUA learning library
12 --- (c) Tim Menzies 2022, BSD-2
13 --- https://menzies.us/l5
14 --- Share and enjoy
15 ---
16 ---
17 ---
18 ---
19 ---
20 ---
21 ---
22 ---
23 ---
24 ---
25 ---
26 local b4={}; for k, _ in pairs(_ENV) do b4[k]=k end
27 local the,help={},{}
28
29 lua l5.lua [OPTIONS]
30 L5 == a very little LUA learning lab
31
32 OPTIONS (inference):
33   -boot -b P #bootstrap samples
34   -cohen -c F cohen's small effect size
35   -cliffs -C F threshold on Cliff's delta
36   -far -F F look no further than "far"
37   -keep -k items to keep in a number
38   -leaves -l leaf size
39   -p -p P distance calcs coefficient
40   -seed -s P random number seed
41   -some -s look only at "some" items
42
43 OPTIONS (housekeeping):
44   -dump -d on error, exit+ stacktrace
45   -file -f S where to get data
46   -help -h show help
47   -rnd -r S format string
48   -todo -t S start-up action
49
50 ]]
51
52 Copyright 2022, Tim Menzies
53
54 Redistribution and use in source and binary forms, with or without
55 modification, are permitted provided that the following conditions
56 are met:
57
58 1. Redistributions of source code must retain the above copyright
59 notice, this list of conditions and the following disclaimer.
60
61 2. Redistributions in binary form must reproduce the above copyright
62 notice, this list of conditions and the following disclaimer in the
63 documentation and/or other materials provided with the distribution.
64
65 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
66 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
67 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
68 FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
69 COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
70 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
71 BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
72 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
73 CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
74 LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
75 ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
76 POSSIBILITY OF SUCH DAMAGE. --]]
77
78 -----
79 -- ## Coding Conventions
80 --
81 -- - All config options in "the" (which is generated by parsing the help text)
82 -- - Line width = 80
83 -- - when you can, write functions down on one line
84 -- - "if" not "self" (so we can fit more on each line)
85 -- - if something holds a list of thing, name the holding variable "all"
86 -- - no inheritance
87 -- - only define a method if that is for polymorphism
88 -- - all config items into a global "the" variable
89 -- - all the test cases (or demos) are "function Demo.xxx".
90 -- - If test case assertion crashed, add "1" to Demo.fails
91 -- - On exit return the value of Demo.fails as the exit status
92 -- - random seed reset so carefully, just once, at the end of the code.
93 -- - usually, no line with just "end" on it

```

```

94 -----
95 ---
96 --- DATA
97 ---
98 --- This code reads date from csv files (where "?" denotes "missing value").
99 local is={}
100 function is.missing(x) return x=="?" end
101
102 --- The names on row1 of that file define the role of that column.
103 --- Names in row1 ending with ":" are to be ignored
104 function is.skip(x) return x:find":" end
105
106 --- Names in row1 starting in upper case are numbers
107 function is.num(x) return x:find"[A-Z]" end
108
109 --- Names in row1 ending with "!" are classes.
110 function is.class(x) return x:find"!" end
111
112 --- Names in row1 ending with "-" are objectives to be minimized.
113 function is.less(x) return x:find"-" end
114
115 --- Names in row1 ending with "+" are objectives to be maximized.
116 function is.more(x) return x:find"+" end
117
118 --- Objectives or classes are dependent variables.
119 function is.dependent(x) return is.more(x) or is.less(x) or is.class(x) end
120
121 --- For example, in this data file, we will ignore column 3 (Hp:),
122 --- try to minimize weight (Lbs-) and maximize acceleration and
123 --- miles per hour (Acc+, Mpg+). Also, with one exception (origin),
124 --- everything is numeric. Finally, there are some missing values on
125 --- lines 3 and lines 7.
126
127 ---
128 --- Clnrds, Weight, Hp:, Lbs-, Acc+, Model, origin, Mpg+
129 --- 8, 304.0, 193, 4732, 18.5, 70, 1, 10
130 --- 8, ?, 215, 4615, 14, 70, 1, 10
131 --- 4, 85, 70, 2070, 18.6, 78, 3, 40
132 --- 4, 85, 65, 2110, 19.2, 80, 3, 40
133 --- 4, 85, ?, 1835, 17.3, 80, 2, 40
134 --- 4, 98, 76, 2144, 14.7, 80, 2, 40
135
136 -----
137 ---
138 --- OBJECTS
139 ---
140 local as = setmetatable
141 local function obj( t )
142   t=({__tostring=o}; t.__index=t
143   return as(t, {__call=function(_,...) return t.new(_,...) end}) end
144
145 local Sym = obj() -- Where to summarize symbols
146 function Sym:new(at,s) return as({
147   is="Sym", -- type
148   at=at or 0, -- column index
149   name=s or "", -- column name
150   n=0, -- number of items summarized in this column
151   all={}, -- all[x] = n means we've seen "n" repeats of "x"
152   most=0, -- count of the most frequently seen symbol
153   mode=nil -- the most commonly seen letter
154 }, Sym) end
155
156 local Num = obj() -- Where to summarize numbers
157 function Num:new(at,s) return as({
158   is="Num", -- type
159   at=at or 0, -- column index
160   name=s or "", -- column name
161   n=0, -- number of items summarizes in this column
162   mu=0, -- mean (updated incrementally)
163   m2=0, -- second moment (updated incrementally)
164   sd=0, -- standard deviation
165   ok=false, -- true if "all" is sorted
166   all={}, -- a sample of items seen so far
167   lo=1E31, -- lowest number seen; initially, big so 1st num sends it low
168   hi=-1E31, -- highest number seen; initially, small so 2st num sends it hi
169   w=is.less(s or "") and -1 or 1 -- "-1"= minimize and "1"= maximize
170 }, Num) end
171
172 local Egs = obj() -- Where to store examples, summarized into Syms or Nums
173 function Egs:new(names, i,col,here) i=as({
174   is="Egs", -- type
175   all={}, -- all the rows
176   names=names, -- list of name
177   cols={}, -- list of all columns (Nums or Syms)
178   x={}, -- independent columns (nothing marked as "skip")
179   y={}, -- dependent columns (nothing marked as "skip")
180   class=nil -- classes
181 },Egs)
182 for at,name in pairs(names) do
183   col = (is.num(name) and Num or Sym) (at,name)
184   i.cols[i+1].cols = col
185   here = is.dependent(name) and i.y or i.x
186   if not is.skip(name) then
187     here[i+1].here = col
188     if is.class(name) then i.class=col end end end
189   return i end
190
191 ---
192 --- cloning
193 ---
194 function Num.clone(i) return Num(i.at, i.name) end
195 function Sym.clone(i) return Sym(i.at, i.name) end
196
197 local data
198 function Egs.clone(i,rows, copy)
199   copy = Egs(i.names)
200   for _,row in pairs(rows or {}) do data(copy,row) end
201   return copy end

```

```

203 -----
204 --- MISC TOOLS
205 ---
206 ---
207 ---
208 local r = math.random
209 local fmt = string.format
210 local unpack = table.unpack
211 local function push(t,x) table.insert(t,x); return x end
212 ---
213 ---
214 ---
215 local thing,things,file2things
216 function thing(x)
217   x = x:match("^%s*(-)%s*$")
218   if x=="true" then return true elseif x=="false" then return false end
219   return tonumber(x) or x end
220 ---
221 function things(x,sep, t)
222   t={}; for y in x:gmatch(sep or "([^\s]+)") do t[1+#t]=thing(y) end
223   return t end
224 ---
225 function file2things(file, x)
226   file = io.input(file)
227   return function()
228     x=io.read();
229     if x then return things(x) else io.close(file) end end end
230 ---
231 ---
232 ---
233 ---
234 local last,per,any,many
235 function last(a) return a[ #a ] end
236 function per(a,p) return a[ (p*#a)//1 ] end
237 function any(a) return a[ math.random(#a) ] end
238 function many(a,n, u) u={}; for j=1,n do push(u,any(a)) end; return u end
239 ---
240 ---
241 ---
242 local firsts,sort,map,slots
243 function firsts(a,b) return a[1] < b[1] end
244 function sort(t,f) table.sort(t,f); return t end
245 function map(t,f, u) u={};for k,v in pairs(t) do push(u,f(v)) end; return u end
246 function slots(t, u,s)
247   u={}
248   for k,v in pairs(t) do s=tostring(k);if s:sub(1,1)~="_" then push(u,k) end end
249   return sort(u) end
250 ---
251 ---
252 ---
253 ---
254 local oo,o, rnd, rnds
255 function oo(t) print(o(t)) end
256 function o(t,seen, key,xseen,u)
257   seen = seen or {}
258   if type(t)~="table" then return tostring(t) end
259   if seen[t] then return "..." end
260   seen[t] = t
261   key = function(k) return fmt(":%s %s",k,o(t[k],seen)) end
262   xseen = function(x) return o(x,seen) end
263   u = #t>0 and map(t,xseen) or map(slots(t),key)
264   return (t.is or "")..'{'..table.concat(u,"")..'}' end
265 ---
266 function rnds(t,f) return map(t, function(x) return rnd(x,f) end) end
267 function rnd(x,f)
268   return fmt(type(x)=="number" and (x~x//1 and f or the.rnd) or "%s",x) end
269 ---
270 ---
271 ---
272 local Demo, ok = {fails=0}
273 function ok(test,msg)
274   print(test and "PASS: "or "FAIL: ",msg or "")
275   if not test then
276     Demo.fails=Demo.fails+1
277     if the.dump then assert(test,msg) end end end
278 ---
279 function Demo.main(todo,seed)
280   for k,one in pairs(todo=="all" and slots(Demo) or {todo}) do
281     if k ~= "main" and type(Demo[one]) == "function" then
282       math.randomseed(seed)
283       Demo[one]() end end
284   for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
285   return Demo.fails end
286 ---
287 ---
288 ---
289 ---
290 local function settings(txt, d)
291   d={}
292   txt:gsub("\n ([-](^%s+))([%s]+(-[^%s+)]^)\n)%s%([%s]+)",
293   function(long,key,short,x)
294     for n,flag in ipairs(arg) do
295       if flag==short or flag==long then
296         x = x=="false" and true or x=="true" and "false" or arg[n+1] end end
297       if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
298         d[key] = tonumber(x) or x end end)
299   if d.help then print(txt) end
300   return d end
301 -----
302 --- USE CASES
303 ---
304 ---
305 ---
306 ---
307 ---
308 ---
309 ---
310 local add
311 function add(i,x, inc)
312   inc = inc or 1
313   if not is.missing(x) then
314     i.n = i.n + inc
315     i:internalAdd(x,inc) end
316   return x end
317 ---
318 function Sym.internalAdd(i,x,inc)
319   i.all[x] = inc + (i.all[x] or 0)
320   if i.all[x] > i.most then i.most, i.mode = i.all[x], x end end
321 ---
322 function Num.internalAdd(i,x,inc, d)
323   for j=1,inc do
324     d = x - i.mu
325     i.mu = i.mu + d/i.n
326     i.m2 = i.m2 + d*(x - i.mu)
327     i.sd = (i.m2<0 or i.n<2) and 0 or ((i.m2/(i.n-1))^0.5)
328     i.lo = math.min(x, i.lo)
329     i.hi = math.max(x, i.hi)
330     if #i.all < the.keep then i.ok=false; push(i.all,x)
331     elseif r() < they.keep/i.n then i.ok=false; i.all[r(#i.all)]=x end end end
332 ---
333 function Num.sorted(i)
334   if not i.ok then i.all = sort(i.all) end
335   i.ok=true
336   return i.all end
337 ---
338 ---
339 ---
340 local file2Egs -- not "local data" (since defined above)
341 function data(i,row)
342   push(i.all, row)
343   for _,col in pairs(i.cols) do add(col, row[col.at]) end
344   return i end
345 ---
346 function file2Egs(file, i)
347   for row in file2things(file) do
348     if i then data(i,row) else i = Egs(row) end end
349   return i end
350 ---
351 ---
352 ---
353 local mids
354 function mids(i,rows,cols) return i:clone(rows):mid(cols) end
355 ---
356 function Egs.mid(i,cols)
357   return map(cols or i.y,function(col) return col:mid(i) end) end
358 ---
359 function Sym.mid(i) return i.mode end
360 function Num.mid(i) return i.mu end
361 ---
362 function Num.div(i) return i.sd end
363 function Sym.div(i, e)
364   e=0; for _,n in pairs(i.all) do e=e + n/i.n*math.log(n/i.n,2) end
365   return -e end
366 ---
367 ---
368 ---
369 local far,furthest,neighbors,dist
370 function far( i,r1,rows,far)
371   return per(neighbors(i,r1,rows),far or the.far)[2] end
372 ---
373 function furthest( i,r1,rows)
374   return last(neighbors(i,r1,rows))[2] end
375 ---
376 function neighbors(i,r1,rows)
377   return sort(map(rows, function(r2) return {dist(i,r1,r2),r2} end),firsts) end
378 ---
379 function dist(i,row1,row2, d,n,a,b,inc)
380   d,n = 0,0
381   for _,col in pairs(i.x) do
382     a,b = row1[col.at], row2[col.at]
383     inc = is.missing(a) and is.missing(b) and 1 or col:dist1(a,b)
384     d = d + inc*the.p
385     n = n + 1 end
386   return (d/n)^(1/the.p) end
387 ---
388 function Sym.dist1(i,a,b) return a==b and 0 or 1 end
389 ---
390 function Num.dist1(i,a,b)
391   if is.missing(a) then b=i:norm(b); a=b<.5 and 1 or 0
392   elseif is.missing(b) then a=i:norm(a); b=a<.5 and 1 or 0
393   else a,b = i:norm(a), i:norm(b) end
394   return math.abs(a - b) end
395 ---
396 function Num.norm(i,x)
397   return i.hi - i.lo < 1E-32 and 0 or (x - i.lo)/(i.hi - i.lo) end
398 ---
399 ---
400 ---
401 local half, cluster, clusters
402 function half(i, rows, project,row,some,left,right,lefts,rights,c,mid)
403   function project(row,a,b)
404     a = dist(i,left,row)
405     b = dist(i,right,row)
406     return ((a^2 + c^2 - b^2)/(2*c), row)
407   end
408   some = many(rows, the.some)
409   left = furthest(i,any(some), some)
410   right = furthest(i,left, some)
411   c = dist(i,left,right)
412   lefts,rights = {},{}
413   for n,projection in pairs(sort(map(rows,project),firsts)) do
414     if n==#rows//2 then mid=row end
415     push(n <= #rows//2 and lefts or rights, projection[2]) end
416   return lefts, rights, left, right, mid, c end
417 ---
418 function cluster(i,rows, here,lefts,rights)
419   rows = rows or i.all
420   here = {all=rows}
421   if #rows >= 2* (#i.all)^the.leaves then
422     lefts, rights, here.left, here.right, here.mid = half(i, rows)
423     if #lefts < #rows then
424       here.lefts = cluster(i,lefts)
425       here.rights = cluster(i,rights) end end
426   return here end
427 ---
428 function clusters(i,format,t,pre, front)
429   if t then
430     pre=pre or ""
431     front = fmt("%s%s",pre,#t.all)
432     if not t.lefts and not t.rights then
433       print(fmt("%%-20s%",front, o(rnds(mids(i,t.all),format))))
434     else
435       print(front)
436       clusters(i,format,t.lefts, "|".. pre)
437     end
438   end
439 ---

```

```
clusters(i,format,t.rights,"|.. pre) end end end
```

```

438 ---
439 ---
440
441 local merge,merged,spans,bestSpan
442 function Sym.spans(i, j)
443   local xys,all,one,last,x,y,n = {}, {}
444   for x,n in pairs(i.all) do push(xys, {x,"lefts",n}) end
445   for x,n in pairs(j.all) do push(xys, {x,"rights",n}) end
446   for _,tmp in ipairs(sort(xys,firsts)) do
447     x,y,n = unpack(tmp)
448     if x ~= last then
449       last = x
450       one = push(all, {lo=x, hi=x, all=Sym(i.at,i.name)}) end
451     add(one.all, y, n) end
452   return all end
453
454 function Num.spans(i, j)
455   local xys,all,lo,hi,gap,one,x,y,n = {}, {}
456   lo,hi = math.min(i.lo, j.lo), math.max(i.hi, j.hi)
457   gap = (hi - lo) / (6/the.cohen)
458   for _,n in pairs(i.all) do push(xys, {n,"lefts",1}) end
459   for _,n in pairs(j.all) do push(xys, {n,"rights",1}) end
460   one = {lo=lo, hi=hi, all=Sym(i.at,i.name)}
461   all = {one}
462   for _,tmp in ipairs(sort(xys,firsts)) do
463     x,y,n = unpack(tmp)
464     if one.hi - one.lo > gap
465       then one = push(all, {lo=one.hi, hi=x, all=one.all:clone()}) end
466     one.hi = x
467     add(one.all, y, n) end
468   all = merge(all)
469   all[1].lo = -math.huge
470   all[#all].hi = math.huge
471   return all end
472
473 function merge(b4, j,n,now,a,b,both)
474   j, n, now = 0, #b4, {}
475   while j < #b4 do
476     j = j+1
477     a, b = b4[j], b4[j+1]
478     if b then
479       both = a.all:merged(b.all)
480       if both
481         then a = {lo=a.lo, hi=b.hi, all=both}
482         j = j + 1 end end
483     push(now,a) end
484   return #now == #b4 and b4 or merge(now) end
485
486 function Sym.merge(i,j, k)
487   k = i:clone()
488   for x,n in pairs(i.all) do add(k,x,n) end
489   for x,n in pairs(j.all) do add(k,x,n) end
490   return k end
491
492 function Sym.merged(i,j, k,ei,ej,ek)
493   k = i:merge(j)
494   ei, ej, ek = i:div(), j:div(), k:div()
495   if ek*.99 <= (i.n*ei + j.n*ej)/k.n then return k end end
496
497 function spans(egs1,egs2, spans,tmp,coll,col2)
498   spans = {}
499   for c,coll in pairs(egs1.x) do
500     col2 = egs2.x[c]
501     tmp = coll:spans(col2)
502     if #tmp> 1 then
503       for _,one in pairs(tmp) do push(spans,one) end end end
504   return spans end
505
506 function bestSpan(spans)
507   local divs,ns,n,div,stats,dist2heaven = Num(), Num()
508   function dist2heaven(s) return {(1 - n(s))^2 + (0 - div(s))^2^.5,s} end
509   function div(s) return divs:norm( s.all:div() ) end
510   function n(s) return ns:norm( s.all.n ) end
511   for _,s in pairs(spans) do
512     add(divs, s.all:div())
513     add(ns, s.all.n) end
514   return sort(map(spans, dist2heaven), firsts)[1][2] end
515
516 ---
517 ---
518
519 local xplain,xplains,selects,spanShow
520 function xplain(i,rows,used,
521   stop,here,left,right,lefts0,rights0,lefts1,rights1)
522   used=used or {}
523   rows = rows or i.all
524   here = {all=rows}
525   stop = (#i.all)^the.leaves
526   if #rows >= 2*stop then
527     lefts0, rights0, here.left, here.right, here.mid, here.c = half(i, rows)
528     if #lefts0 < #rows then
529       here.selector = bestSpan(spans(i:clone(lefts0),i:clone(rights0)))
530       push(used, {here.selector.all.name, here.selector.lo, here.selector.hi})
531       lefts1,rights1 = {}, {}
532       for _,row in pairs(rows) do
533         push(selects(here.selector, row) and lefts1 or rights1, row) end
534       if #lefts1 > stop then here.lefts = xplain(i,lefts1,used) end
535       if #rights1 > stop then here.rights = xplain(i,rights1,used) end end end
536   return here end
537
538 function xplains(i,format,t,pre,how, sel,front)
539   pre, how = pre or "", how or ""
540   if t then
541     pre=pre or ""
542     front = fmt("%s%s%s",pre,how, #t.all, t.c and rnd(t.c) or "")
543     if t.lefts and t.rights then print(fmt("%-35s",front)) else
544       print(fmt("%-35s",front, o(rnds(mids(i,t.all),format))))
545     end
546     sel = t.selector
547     xplains(i,format,t.lefts, "|.. pre, spanShow(sel..":")
548     xplains(i,format,t.rights, "|.. pre, spanShow(sel,true) ..":") end end
549
550 function selects(span,row, lo,hi,at,x)
551   lo, hi, at = span.lo, span.hi, span.all.at
552   x = row[at]
553   if is.missing(x) then return true end
554   if lo==hi then return x==lo else return lo <= x and x < hi end end
555
556 function spanShow(span, negative, hi,lo,x,big)
557   if not span then return "" end
558   lo, hi, x, big = span.lo, span.hi, span.all.name, math.huge
559   if not negative
560     then if lo == hi then return fmt("%s== %s",x,lo) end
561           if hi == big then return fmt("%s>= %s",x,lo) end
562           if lo == -big then return fmt("%s< %s",x,hi) end
563           return fmt("%s<= %s< %s",lo,x,hi)
564         else if lo == hi then return fmt("%s!= %s",x,lo) end
565               if hi == big then return fmt("%s< %s",x,lo) end
566               if lo == -big then return fmt("%s>= %s",x,hi) end
567               return fmt("%s< %s and %s>= %s", x,lo,x,hi) end end

```

```

568 ---  _||_||_||_
569 ---
570
571 -- function Num.same(i,j, xs,ys,      lt,gt)
572 --   lt,gt = 0, 0
573 --   for _,x in pairs(i.all) do
574 --     for _,y in pairs(i.all) do
575 --       if y > x then gt = gt + 1 end
576 --       if y < x then lt = lt + 1 end end end
577 --   return math.abs(gt - lt)/(#xs * #ys) <= the.cliffs end
578 --
579 -- ## Significance
580 -- Non parametric "significance" test (i.e. is it possible to
581 -- distinguish if an item belongs to one population of
582 -- another). Two populations are the same if no difference can be
583 -- seen in numerous samples from those populations.
584 -- Warning: very
585 -- slow for large populations. Consider sub-sampling for large
586 -- lists. Also, test the effect size (and maybe shortcut the
587 -- test) before applying this test. From p220 to 223 of the
588 -- Efron text "introduction to the bootstrap".
589 -- https://bit.ly/3iSjz8B Typically, conf=0.05 and b is 100s to
590 -- 1000s.
591 -- Translate both samples so that they have mean x,
592 -- The re-sample each population separately.
593 -- function bootstrap(y0,z0,my)
594 --   local x,y,z,xmu,ymu,zmu,yhat,zhat,tobs,ns, bootstraps, confidence
595 --   bootstraps = my and my.bootstrap or 512
596 --   confidence = my and my.conf or .05
597 --   x, y, z, yhat, zhat = Num.new(), Num.new(), {}, {}
598 --   for _,y1 in pairs(y0) do x:summarize(y1); y:summarize(y1) end
599 --   for _,z1 in pairs(z0) do x:summarize(z1); z:summarize(z1) end
600 --   xmu, ymu, zmu = x.mu, y.mu, z.mu
601 --   for _,y1 in pairs(y0) do yhat[1+#yhat] = y1 - ymu + xmu end
602 --   for _,z1 in pairs(z0) do zhat[1+#zhat] = z1 - zmu + xmu end
603 --   tobs = y:delta(z)
604 --   n = 0
605 --   for _ = 1,bootstraps do
606 --     if adds(samples(yhat)):delta(adds(samples(zhat))) > tobs
607 --       then n = n + 1 end end
608 --   return n / bootstraps >= conf end
609 --
610 -- function scottKnot(nums,the,      all,cohen)
611 --   local mid = function (z) return z.some:mid()
612 --   end -----
613 --   local function summary(i,j,      out)
614 --     out = copy( nums[i] )
615 --     for k = i+1, j do out = out:merge(nums[k]) end
616 --     return out
617 --   end -----
618 --   local function div(lo,hi,rank,b4,      cut,best,l,l1,r,r1,now)
619 --     best = 0
620 --     for j = lo,hi do
621 --       if j < hi then
622 --         l = summary(lo, j)
623 --         r = summary(j+1, hi)
624 --         now = (l.n*(mid(l) - mid(b4))^2 + r.n*(mid(r) - mid(b4))^2
625 --           ) / (l.n + r.n)
626 --         if now > best then
627 --           if math.abs(mid(l) - mid(r)) >= cohen then
628 --             cut, best, l1, r1 = j, now, copy(l), copy(r)
629 --           end end end end
630 --         if cut and not l1:same(r1,the) then
631 --           rank = div(lo,      cut, rank, l1) + 1
632 --           rank = div(cut+1, hi, rank, r1)
633 --         else
634 --           for i = lo,hi do nums[i].rank = rank end end
635 --         return rank
636 --       end -----
637 --       table.sort(nums, function(x,y) return mid(x) < mid(y) end)
638 --       all = summary(1,#nums)
639 --       cohen = all.sd * the.iota
640 --       div(1, #nums, 1, all)
641 --       return nums end
642 --

```

```

643 -----
644 ---  MATH
645 ---
646 ---
647 -- function Demo.the() oo(the) end
648 --
649 -- function Demo.many(a)
650 --   a={1,2,3,4,5,6,7,8,9,10}; ok("{1023}" == o(many(a,3)), "manys") end
651 --
652 -- function Demo.egs()
653 --   ok(5140==file2Egs(the.file).y[1].hi,"reading") end
654 --
655 -- function Demo.dist(i)
656 --   i = file2Egs(the.file)
657 --   for n,row in pairs(i.all) do print(n,dist(i, i.all[1], row)) end end
658 --
659 -- function Demo.far( i,j,row1,row2,row3,d3,d9)
660 --   i = file2Egs(the.file)
661 --   for j=1,10 do
662 --     row1 = any(i.all)
663 --     row2 = far(i,row1, i.all, .9)
664 --     d9 = dist(i,row1,row2)
665 --     row3 = far(i,row1, i.all, .3)
666 --     d3 = dist(i,row1,row3)
667 --     ok(d3 < d9, "closer far") end end
668 --
669 -- function Demo.half( i, lefts, rights)
670 --   i = file2Egs(the.file)
671 --   lefts, rights = half(i, i.all)
672 --   oo(mids(i, lefts))
673 --   oo(mids(i, rights))
674 -- end
675 --
676 -- function Demo.cluster( i)
677 --   i = file2Egs(the.file)
678 --   clusters(i, "%0f", cluster(i)) end
679 --
680 -- function Demo.spans( i, lefts, rights)
681 --   i = file2Egs(the.file)
682 --   lefts, rights = half(i, i.all)
683 --   oo(bestSpan(spans(i:clone(lefts), i:clone(rights)))) end
684 --
685 -- function Demo.xplain( i,j,tmp, lefts, rights, used)
686 --   i = file2Egs(the.file)
687 --   used={}
688 --   xplains(i, "%0f", xplain(i, i.all, used))
689 --   map(sort(used, function(a,b)
690 --     return ((a[1] < b[1]) or
691 --       (a[1]==b[1] and a[2] < b[2]) or
692 --       (a[1]==b[1] and a[2]==b[2] and a[3] < b[3]))end),oo) end
693 --
694 --
695 -- the = settings(help)
696 -- Demo.main(the.todo, the.seed)

```