

```

1  -----
2  ---
3  ---
4  ---
5  ---
6  ---
7  ---
8  ---
9  ---
10 ---
11 ---
12 ---
13 ---
14 ---
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          DEFAULT
34 -cohen -c F cohen's small effect size    256
35 -cliffs -C F threshold on Cliff's delta  .35
36 -far -F F look no further than "far"     .147
37 -keep -k items to keep in a number      .9
38 -leaves -l leaf size                    512
39 -conf -n F confidence for stats tests    .5
40 -p -p P distance calcs coefficient       .05
41 -seed -S P random number seed          10019
42 -some -s look only at "some" items      512
43
44 OPTIONS (housekeeping):
45 -dump -d on error, exit+ stacktrace      false
46 -file -f S where to get data             ./etc/data/auto93.csv
47 -help -h show help                      false
48 -rnd -r S format string                  %5.2f
49 -todo -t S start-up action               nothing
50 ]]
51 -----
52 --[[
53 Copyright 2022, Tim Menzies
54
55 Redistribution and use in source and binary forms, with or without
56 modification, are permitted provided that the following conditions
57 are met:
58
59 1. Redistributions of source code must retain the above copyright
60 notice, this list of conditions and the following disclaimer.
61
62 2. Redistributions in binary form must reproduce the above copyright
63 notice, this list of conditions and the following disclaimer in the
64 documentation and/or other materials provided with the distribution.
65
66 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
67 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
68 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
69 FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
70 COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
71 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
72 BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
73 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
74 CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
75 LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
76 ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
77 POSSIBILITY OF SUCH DAMAGE. --]]
78
79 -----
80 -- ## Coding Conventions
81 --
82 -- - All config options in "the" (which is generated by parsing the help text)
83 -- - LOTS OF SHORT FUNCTIONS
84 -- - Line width = 80
85 -- - when you can, write functions down on one line
86 -- - "i" not "self" (so we can fit more on each line)
87 -- - for loop index variables, do not use i. try j,k instead.
88 -- - if something holds a list of thing, name the holding variable "all"
89 -- - no inheritance
90 -- - only define a method if that is for polymorphism
91 -- - all config items into a global "the" variable
92 -- - all the test cases (or demos) are "function Demo.xxx".
93 -- - If test case assertion crashed, add "1" to Demo.fails
94 -- - On exit return the value of Demo.fails as the exit status
95 -- - random seed reset so carefully, just once, at the end of the code.
96 -- - usually, no line with just "end" on it

```

```

97  -----
98  ---
99  ---
100 ---
101 ---
102 ---
103 ---
104 ---
105 ---
106 ---
107 ---
108 ---
109 ---
110 ---
111 ---
112 ---
113 ---
114 ---
115 ---
116 ---
117 ---
118 ---
119 ---
120 ---
121 ---
122 ---
123 ---
124 ---
125 ---
126 ---
127 ---
128 ---
129 ---
130 ---
131 ---
132 ---
133 ---
134 ---
135 ---
136 ---
137 ---
138 ---
139 ---
140 ---
141 ---
142 ---
143 ---
144 ---
145 ---
146 ---
147 ---
148 ---
149 ---
150 ---
151 ---
152 ---
153 ---
154 ---
155 ---
156 ---
157 ---
158 ---
159 ---
160 ---
161 ---
162 ---
163 ---
164 ---
165 ---
166 ---
167 ---
168 ---
169 ---
170 ---
171 ---
172 ---
173 ---
174 ---
175 ---
176 ---
177 ---
178 ---
179 ---
180 ---
181 ---
182 ---
183 ---
184 ---
185 ---
186 ---
187 ---
188 ---
189 ---
190 ---
191 ---
192 ---
193 ---
194 ---
195 ---
196 ---
197 ---
198 ---
199 ---
200 ---
201 ---
202 ---
203 ---
204 ---
205 ---

```

```

206 -----
207 --- MISC TOOLS
208 ---
209 ---
210 ---
211 local r = math.random
212 local fmt = string.format
213 local unpack = table.unpack
214 local function push(t,x) table.insert(t,x); return x end
215 ---
216 ---
217 ---
218 local thing,things,file2things
219 function thing(x)
220 x = x:match("%s*(-)%s*$")
221 if x=="true" then return true elseif x=="false" then return false end
222 return tonumber(x) or x end
223 ---
224 function things(x,sep, t)
225 t={}; for y in x:gmatch(sep or "([^\s]+)") do t[1+#t]=thing(y) end
226 return t end
227 ---
228 function file2things(file, x)
229 file = io.input(file)
230 return function()
231 x=io.read();
232 if x then return things(x) else io.close(file) end end end
233 ---
234 ---
235 ---
236 ---
237 local last,per,any,many
238 function last(a) return a[#a] end
239 function per(a,p) return a[(p*#a)//1] end
240 function any(a) return a[math.random(#a)] end
241 function many(a,n, u) u={}; for j=1,n do push(u,any(a)) end; return u end
242 ---
243 ---
244 ---
245 local firsts,sort,map,slots,copy
246 function firsts(a,b) return a[1] < b[1] end
247 function sort(t,f) table.sort(t,f); return t end
248 function map(t,f, u) u={};for k,v in pairs(t) do push(u,f(v)) end; return u end
249 function slots(t, u,s)
250 u={}
251 for k,v in pairs(t) do s=tosstring(k);if s:sub(1,1)~="_" then push(u,k) end end
252 return sort(u) end
253 ---
254 function copy(t, u)
255 if type(t)~="table" then return t end
256 u={}; for k,v in pairs(t) do u[copy(k)]=copy(v) end
257 return setmetatable(u, getmetatable(t)) end
258 ---
259 ---
260 ---
261 ---
262 ---
263 local oo,o, rnd, rnds
264 function oo(t) print(o(t)) end
265 function o(t,seen, key,xseen,u)
266 seen = seen or {}
267 if type(t)~="table" then return tostring(t) end
268 if seen[t] then return "..." end
269 seen[t] = t
270 key = function(k) return fmt("%.5s %s",k,o(t[k],seen)) end
271 xseen = function(x) return o(x,seen) end
272 u = #t>0 and map(t,xseen) or map(slots(t),key)
273 return (t.is or "")..'{'..table.concat(u,"")..'}' end
274 ---
275 function rnds(t,f) return map(t, function(x) return rnd(x,f) end) end
276 function rnd(x,f)
277 return fmt(type(x)=="number" and (x~=x//1 and f or the.rnd) or "%s",x) end
278 ---
279 ---
280 ---
281 local Demo, ok = {fails=0}
282 function ok(test,msg)
283 print(test and "PASS:" or "FAIL:",msg or "")
284 if not test then
285 Demo.fails=Demo.fails+1
286 if the.dump then assert(test,msg) end end end
287 ---
288 function Demo.main(todo,seed)
289 for k,one in pairs(todo=="all" and slots(Demo) or {todo}) do
290 if k ~="main" and type(Demo[one]) == "function" then
291 math.randomseed(seed)
292 Demo[one]() end end
293 for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
294 return Demo.fails end
295 ---
296 ---
297 ---
298 ---
299 local function settings(txt, d)
300 d={}
301 txt:gsub("\n ([-]|([^\s]+))([^\s]+)([^\s]+)[^\n]*%s([^\s]+)",
302 function(long,key,short,x)
303 for n,flag in ipairs(arg) do
304 if flag==short or flag==long then
305 x = x=="false" and true or x=="true" and "false" or arg[n+1] end end
306 if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
307 d[key] = tonumber(x) or x end end
308 if d.help then print(txt) end
309 return d end

```

```

310 -----
311 --- USE CASES
312 ---
313 ---
314 ---
315 ---
316 --- update cols
317 ---
318 ---
319 local add
320 function add(i,x, inc)
321 inc = inc or 1
322 if not is.missing(x) then
323 i.n = i.n + inc
324 i:internalAdd(x,inc) end
325 return x end
326 ---
327 function Sym.internalAdd(i,x,inc)
328 i.all[x] = inc + (i.all[x] or 0)
329 if i.all[x] > i.most then i.most, i.mode = i.all[x], x end end
330 ---
331 function Num.internalAdd(i,x,inc, d)
332 for j=1,inc do
333 d = x - i.mu
334 i.mu = i.mu + d/i.n
335 i.m2 = i.m2 + d*(x - i.mu)
336 i.sd = (i.m2<0 or i.n<2) and 0 or ((i.m2/(i.n-1))^0.5)
337 i.lo = math.min(x, i.lo)
338 i.hi = math.max(x, i.hi)
339 if i.all < the.keep then i.ok=false; push(i.all,x)
340 elseif r() < the.keep/i.n then i.ok=false; i.all[r(#i.all)]=x end end end
341 ---
342 function Num.sorted(i)
343 if not i.ok then i.all = sort(i.all) end
344 i.ok=true
345 return i.all end
346 ---
347 ---
348 ---
349 local file2Egs -- not "local data" (since defined above)
350 function data(i,row)
351 push(i.all, row)
352 for _,col in pairs(i.cols) do add(col, row[col.at]) end
353 return i end
354 ---
355 function file2Egs(file, i)
356 for row in file2things(file) do
357 if i then data(i,row) else i = Egs(row) end end
358 return i end
359 ---
360 ---
361 ---
362 local mids
363 function mids(i,rows,cols) return i:clone(rows):mid(cols) end
364 ---
365 function Egs.mid(i,cols)
366 return map(cols or i.y,function(col) return col:mid(i) end) end
367 ---
368 function Sym.mid(i) return i.mode end
369 function Num.mid(i) return i.mu end
370 ---
371 function Num.div(i) return i.sd end
372 function Sym.div(i, e)
373 e=0; for _,n in pairs(i.all) do e = e + n/i.n*math.log(n/i.n,2) end
374 return -e end
375 ---
376 ---
377 ---
378 local far,furthest,neighbors,dist
379 function far(i,r1,rows,far)
380 return per(neighbors(i,r1,rows),far or the.far)[2] end
381 ---
382 function furthest(i,r1,rows)
383 return last(neighbors(i,r1,rows))[2] end
384 ---
385 function neighbors(i,r1,rows)
386 return sort(map(rows, function(r2) return {dist(i,r1,r2),r2} end),firsts) end
387 ---
388 function dist(i,row1,row2, d,n,a,b,inc)
389 d,n = 0,0
390 for _,col in pairs(i.x) do
391 a,b = row1[col.at], row2[col.at]
392 inc = is.missing(a) and is.missing(b) and 1 or col:dist1(a,b)
393 d = d + inc*the.p
394 n = n + 1 end
395 return (d/n)^(1/the.p) end
396 ---
397 function Sym.dist1(i,a,b) return a==b and 0 or 1 end
398 ---
399 function Num.dist1(i,a,b)
400 if is.missing(a) then b=i:norm(b); a=b<.5 and 1 or 0
401 elseif is.missing(b) then a=i:norm(a); b=a<.5 and 1 or 0
402 else a,b = i:norm(a), i:norm(b) end
403 return math.abs(a - b) end
404 ---
405 function Num.norm(i,x)
406 return i.hi - i.lo < 1E-32 and 0 or (x - i.lo)/(i.hi - i.lo) end

```

```

407 --- c | | | | | | | |
408 ---
409 local half, cluster, clusters
410 function half(i, rows, project, row, some, left, right, lefts, rights, c, mid)
411 function project(row, a, b)
412 a = dist(i, left, row)
413 b = dist(i, right, row)
414 return {(a^2 + c^2 - b^2)/(2*c), row}
415 end
416 some = many(rows, the.some)
417 left = furthest(i, any(some), some)
418 right = furthest(i, left, some)
419 c = dist(i, left, right)
420 lefts, rights = {}, {}
421 for n, projection in pairs(sort(map(rows, project), firsts)) do
422 if n == #rows//2 then mid = row end
423 push(n <= #rows//2 and lefts or rights, projection[2]) end
424 return lefts, rights, left, right, mid, c end
425
426 function cluster(i, rows, here, lefts, rights)
427 rows = rows or i.all
428 here = {all=rows}
429 if #rows >= 2* (#i.all)^the.leaves then
430 lefts, rights, here.left, here.right, here.mid = half(i, rows)
431 if #lefts < #rows then
432 here.lefts = cluster(i, lefts)
433 here.rights = cluster(i, rights) end end
434 return here end
435
436 function clusters(i, format, t, pre, front)
437 if t then
438 pre = pre or ""
439 front = fmt("%s%s", pre, #t.all)
440 if not t.lefts and not t.rights then
441 print(fmt("%-20s", front, o(rnds(mids(i, t.all), format))))
442 else
443 print(front)
444 clusters(i, format, t.lefts, "|" .. pre)
445 clusters(i, format, t.rights, "|" .. pre) end end end

```

```

447 --- c | | | | | | | |
448 ---
449 local merge, merged, spans, bestSpan
450 function Sym.spans(i, j)
451 local xys, all, one, last, x, y, n = {}, {}
452 for x, n in pairs(i.all) do push(xys, {x, "lefts", n}) end
453 for x, n in pairs(j.all) do push(xys, {x, "rights", n}) end
454 for _, tmp in ipairs(sort(xys, firsts)) do
455 x, y, n = unpack(tmp)
456 if x ~= last then
457 last = x
458 one = push(all, {lo=x, hi=x, all=Sym(i.at, i.name)}) end
459 add(one.all, y, n) end
460 return all end
461
462 function Num.spans(i, j)
463 local xys, all, lo, hi, gap, one, x, y, n = {}, {}
464 lo, hi = math.min(i.lo, j.lo), math.max(i.hi, j.hi)
465 gap = (hi - lo) / (6/the.cohen)
466 for _, n in pairs(i.all) do push(xys, {n, "lefts", 1}) end
467 for _, n in pairs(j.all) do push(xys, {n, "rights", 1}) end
468 one = {lo=lo, hi=hi, all=Sym(i.at, i.name)}
469 all = {one}
470 for _, tmp in ipairs(sort(xys, firsts)) do
471 x, y, n = unpack(tmp)
472 if one.hi - one.lo > gap
473 then one = push(all, {lo=one.hi, hi=x, all=one.all:clone()}) end
474 one.hi = x
475 add(one.all, y, n) end
476 all = merge(all)
477 all[1].lo = -math.huge
478 all[#all].hi = math.huge
479 return all end
480
481 function merge(b4, j, n, now, a, b, both)
482 j, n, now = 0, #b4, {}
483 while j < #b4 do
484 j = j+1
485 a, b = b4[j], b4[j+1]
486 if b then
487 both = a.all:merged(b.all)
488 if both
489 then a = {lo=a.lo, hi=b.hi, all=both}
490 j = j + 1 end end
491 push(now, a) end
492 return #now == #b4 and b4 or merge(now) end
493
494 -- XXX make .merged and function
495 function Num.merge(i, j, k)
496 k = i:clone()
497 for _, x in pairs(i.all) do add(k, x) end
498 for _, x in pairs(j.all) do add(k, x) end
499 return k end
500
501 function Sym.merge(i, j, k)
502 k = i:clone()
503 for x, n in pairs(i.all) do add(k, x, n) end
504 for x, n in pairs(j.all) do add(k, x, n) end
505 return k end
506
507 function Sym.merged(i, j, k, ei, ej, ek)
508 k = i:merge(j)
509 ei, ej, ek = i:div(), j:div(), k:div()
510 if ek*.99 <= (i.n*ei + j.n*ej)/k.n then return k end end
511
512 function spans(egs1, egs2, spans, tmp, coll, col2)
513 spans = {}
514 for c, coll in pairs(egs1.x) do
515 col2 = egs2.x[c]
516 tmp = coll:spans(col2)
517 if #tmp > 1 then
518 for _, one in pairs(tmp) do push(spans, one) end end end
519 return spans end
520
521 function bestSpan(spans)
522 local divs, ns, n, div, stats, dist2heaven = Num(), Num()
523 function dist2heaven(s) return {(1 - n(s))^2 + (0 - div(s))^2^.5, s} end
524 function div(s) return divs:norm(s.all:div()) end
525 function n(s) return ns:norm(s.all.n) end
526 for _, s in pairs(spans) do
527 add(divs, s.all:div())
528 add(ns, s.all.n) end
529 return sort(map(spans, dist2heaven), firsts)[1][2] end
530
531 ---
532 --- c | | | | | | | |
533 ---
534 local xplain, xplans, selects, spanShow
535 function xplain(i, rows, used, stop, here, left, right, lefts0, rights0, lefts1, rights1)
536 used = used or {}
537 rows = rows or i.all
538 here = {all=rows}
539 stop = (#i.all)^the.leaves
540 if #rows >= 2*stop then
541 lefts0, rights0, here.left, here.right, here.mid, here.c = half(i, rows)
542 if #lefts0 < #rows then
543 here.selector = bestSpan(spans(i:clone(lefts0), i:clone(rights0)))
544 push(used, {here.selector.all.name, here.selector.lo, here.selector.hi})
545 lefts1, rights1 = {}, {}
546 for _, row in pairs(rows) do
547 push(selects(here.selector, row) and lefts1 or rights1, row) end
548 if #lefts1 > stop then here.lefts = xplain(i, lefts1, used) end
549 if #rights1 > stop then here.rights = xplain(i, rights1, used) end end end
550 return here end
551
552 function xplans(i, format, t, pre, how, sel, front)
553 pre, how = pre or "", how or ""
554 if t then
555 pre = pre or ""
556 front = fmt("%s%s%s", pre, how, #t.all, t.c and rnd(t.c) or "")
557 if t.lefts and t.rights then print(fmt("%-35s", front)) else
558 print(fmt("%-35s", front, o(rnds(mids(i, t.all), format))))
559 end
560 sel = t.selector
561 xplans(i, format, t.lefts, "|" .. pre, spanShow(sel, true) .. ":")
562 xplans(i, format, t.rights, "|" .. pre, spanShow(sel, true) .. ":") end end
563
564 function selects(span, row, lo, hi, at, x)
565 lo, hi, at = span.lo, span.hi, span.all.at
566 x = row[at]
567 if is.missing(x) then return true end
568 if lo == hi then return x == lo else return lo <= x and x < hi end end
569
570 function spanShow(span, negative, hi, lo, x, big)
571 if not span then return "" end
572 lo, hi, x, big = span.lo, span.hi, span.all.name, math.huge
573 if not negative
574 then if lo == hi then return fmt("%s== %s", x, lo) end
575 if hi == big then return fmt("%s>= %s", x, lo) end
576 if lo == -big then return fmt("%s< %s", x, hi) end
577 return fmt("%s<= %s", lo, x, hi)
578 else if lo == hi then return fmt("%s!= %s", x, lo) end
579 if hi == big then return fmt("%s< %s", x, lo) end
580 if lo == -big then return fmt("%s>= %s", x, hi) end

```

```

583 return fmt ("%s < %s and %s >= %s", x, lo, x, hi) end end
584 ---
585 ---
586 ---
587 local quintiles, smallfx, bootstrap
588 function quintiles(ts, width, nums, out, all, n, m)
589   width = width or 32
590   nums = Num(); for _, t in pairs(ts) do
591     for _, x in pairs(sort(t)) do add(nums, x) end end
592   all, out = nums.all, {}
593   for _, t in pairs(ts) do
594     local s, where = {}
595     where = function(n) return (width*nums:norm(n))/1 end
596     for j = 1, width do s[j] = " " end
597     for j = where(per(t, .1)), where(per(t, .3)) do s[j] = "-" end
598     for j = where(per(t, .7)), where(per(t, .9)) do s[j] = "-" end
599     s[where(per(t, .5))] = "|"
600     push(out, {display=table.concat(s),
601               data = t,
602               pers = map({.1, .3, .5, .7, .9},
603                         function(p) return rnd(per(t, p)) end)}) end
604   return out end
605
606 function smallfx(xs, ys, x, y, lt, gt, n)
607   lt, gt, n = 0, 0, 0
608   if #ys > #xs then xs, ys = ys, xs end
609   for _, x in pairs(xs) do
610     for j = 1, math.min(64, #ys) do
611       y = any(ys)
612       if y < x then lt = lt + 1 end
613       if y > x then gt = gt + 1 end
614       n = n + 1 end end
615   return math.abs(gt - lt) / n <= the.cliffs end
616
617 function bootstrap(y0, z0)
618   local x, y, z, b4, yhat, zhat, bigger
619   local function obs(a, b, c)
620     c = math.abs(a.mu - b.mu)
621     return (a.sd + b.sd) == 0 and c or c / ((x.sd^2/x.n + y.sd^2/y.n)^.5) end
622   local function adds(t, num)
623     num = num or Num(); map(t, function(x) add(num, x) end); return num end
624   y, z = adds(y0), adds(z0)
625   x = adds(y0, adds(z0))
626   b4 = obs(y, z)
627   yhat = map(y.all, function(y1) return y1 - y.mu + x.mu end)
628   zhat = map(z.all, function(z1) return z1 - z.mu + x.mu end)
629   bigger = 0
630   for j = 1, the.boot do
631     if obs( adds(many(yhat, #yhat)), adds(many(zhat, #zhat))) > b4
632     then bigger = bigger + 1/the.boot end end
633   return bigger >= the.conf end
634
635 --- xxx mid has to be per and
636 --- XXXX implement same
637 --- XXXX need tests for stats
638 function scottKnot(nums, all, cohen)
639   local mid = function(z) return z.some:mid() end
640   local function summary(i, j, out)
641     out = copy(nums[i])
642     for k = i+1, j do out = out:merge(nums[k]) end
643     return out
644   end
645   local function div(lo, hi, rank, b4, cut, best, l, ll, r, rl, now)
646     best = 0
647     for j = lo, hi do
648       if j < hi then
649         l = summary(lo, j)
650         r = summary(j+1, hi)
651         now = (l.n*(mid(l) - mid(b4))^2 + r.n*(mid(r) - mid(b4))^2) / (l.n + r.n)
652         if now > best then
653           if math.abs(mid(l) - mid(r)) >= cohen then
654             cut, best, ll, rl = j, now, copy(l), copy(r)
655           end end end
656       if cut and not ll:same(rl, the) then
657         rank = div(lo, cut, rank, ll) + 1
658         rank = div(cut+1, hi, rank, rl)
659       else
660         for i = lo, hi do nums[i].rank = rank end end
661       return rank
662     end
663   end
664   table.sort(nums, function(x, y) return mid(x) < mid(y) end)
665   all = summary(1, #nums)
666   cohen = all.sd * the.cohen
667   div(1, #nums, 1, all)
668   return nums end
669
670
671 -----
672 ---
673 ---
674 ---
675 ---
676 function Demo.the() oo(the) end
677
678 function Demo.many(a)
679   a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; ok("{1023}" == o(many(a, 3)), "manys") end
680
681 local function normal(m, s)
682   local pi, sqrt, cos, log = math.pi, math.sqrt, math.cos, math.log
683   local function z() return sqrt(-2*log(r())) * cos(2*pi*r()) end
684   return m + s*z() end
685
686 function Demo.tiles()
687   local function ns(m, s, r, u)
688     u = {}; for j = 1, r do u[j+1] = normal(m, s) end; return u end
689   local ts = {}
690   local m = 100
691   for mu = 8, 12, .25 do ts[1+#ts] = ns(mu, 5, m) end
692   ts = sort(map(ts, sort), function(a, b) return per(a, .5) < per(b, .5) end)
693   for j, one in pairs(quintiles(ts, 20)) do
694     print(fmt("[%s]", one.display), o(one.pers),
695           smallfx(ts[1], ts[j]),
696           bootstrap(ts[1], ts[j])) end end
697
698 function Demo.stats( t1, t2, inc, n, a, b)
699   for _, n in pairs(20) do --25, 50, 100, 250, 500, 1000 do
700     inc = 1
701     while inc < 3 do
702       print("")
703       t1 = {}; for j = 1, n do push(t1, j*r()) end
704       t2 = {}; for j, x in pairs(t1) do t2[j] = x + inc end
705       a, b = smallfx(t1, t2), bootstrap(t1, t2)
706       for _, x in pairs(quintiles(t1, t2)) do print(rnd(inc), x.display, a, b) end
707       inc = inc * 1.1 end end end
708
709 function Demo.stats1(x)
710   x1 = {0.34, 0.49, 0.51, 0.6}
711   x2 = {0.6, 0.7, 0.8, 0.9}
712   x3 = {0.15, 0.25, 0.4, 0.35}
713   x4 = {0.6, 0.7, 0.8, 0.9}
714   x5 = {0.1, 0.2, 0.3, 0.4}
715   print(bootstrap(x5, x3))
716   print(bootstrap(x3, x1))
717   print(bootstrap(x1, x2))
718   print(bootstrap(x2, x4))
719   end
720
721 function Demo.egs()
722   ok(5140 == file2Egs(the.file).y[1].hi, "reading") end
723
724 function Demo.dist(i)
725   i = file2Egs(the.file)
726   for n, row in pairs(i.all) do print(n, dist(i, i.all[1], row)) end end
727
728 function Demo.far( i, j, row1, row2, row3, d3, d9)
729   i = file2Egs(the.file)
730   for j = 1, 10 do
731     row1 = any(i.all)
732     row2 = far(i, row1, i.all, .9)
733     d9 = dist(i, row1, row2)
734     row3 = far(i, row1, i.all, .3)
735     d3 = dist(i, row1, row3)
736     ok(d3 < d9, "closer far") end end
737
738 function Demo.half( i, lefts, rights)
739   i = file2Egs(the.file)
740   lefts, rights = half(i, i.all)
741   oo(mids(i, lefts))
742   oo(mids(i, rights))
743   end
744
745 function Demo.cluster( i)
746   i = file2Egs(the.file)
747   clusters(i, "%0f", cluster(i)) end
748
749 function Demo.spans( i, lefts, rights)
750   i = file2Egs(the.file)
751   lefts, rights = half(i, i.all)
752   oo(bestSpan(spans(i:clone(lefts), i:clone(rights)))) end
753
754 function Demo.xplain( i, j, tmp, lefts, rights, used)
755   i = file2Egs(the.file)
756   used = {}
757   xplains(i, "%0f", xplain(i, i.all, used))
758   map(sort(used, function(a, b)
759     return ((a[1] < b[1]) or
760            (a[1] == b[1] and a[2] < b[2]) or
761            (a[1] == b[1] and a[2] == b[2] and a[3] < b[3])) end), oo) end
762
763 the = settings(help)
764 Demo.main(the.todo, the.seed)
765
766 -----
767

```