

```

1  _=require"weak"
2  local NUM=_NUM
3  local the,R,rnd,rnds,per  = _the, _R, _rnd, _rnds, _per
4  local fmt,map,co,o,push,sort = _fmt, _map, _co, _o, _push, _sort
5
6  the.conf=0.05
7  the.boot=500
8  the.cliffs=0.147
9
10 local function quintiles(ts,width,  nums,out,n,m)
11   width=width or 32
12   nums=NUM(); for _,t in pairs(ts) do
13     for _,x in pairs(sort(t)) do nums:add(x) end end
14   out = {}
15   for _,t in pairs(ts) do
16     local s, where = {}
17     where = function(n) return (width*nums:norm(n))/1 end
18     for j = 1, width do s[j]=" " end
19     for j = where(per(t,.1)), where(per(t,.3)) do s[j]="-" end
20     for j = where(per(t,.7)), where(per(t,.9)) do s[j]="=" end
21     s[where(per(t,.5))] = " "
22     push(out,(display=table.concat(s),
23             data = t,
24             pers = map({.1,.3,.5,.7,.9},
25                       function(p) return fmt("%6s",rnd(per(t,p)))end))) end
26   return out end
27
28 function smallfx(xs,ys,      x,y,lt,gt,n)
29   lt,gt,n = 0,0,0
30   if #ys > #xs then xs,ys=ys,xs end
31   for _,x in pairs(xs) do
32     for j=1, math.min(64,#ys) do
33       y = any(ys)
34       if y<x then lt=lt+1 end
35       if y>x then gt=gt+1 end
36     n = n+1 end end
37   return math.abs(gt - lt) / n <= the.cliffs end
38
39 function bootstrap(y0,z0,      x,y,z,b4,yhat,zhat,bigger,obs,adds)
40   function obs(a,b,      c)
41     c = math.abs(a.mu - b.mu)
42     return (a.sd + b.sd) == 0 and c or c/((x.sd^2/x.n + y.sd^2/y.n)^.5) end
43   function adds(t, num)
44     num = NUM(); map(t, function(x) num:add(x) end); return num end
45   y,z  = adds(y0), adds(z0)
46   x    = adds(y0, adds(z0))
47   b4   = obs(y,z)
48   yhat = map(y_all, function(y1) return y1 - y.mu + x.mu end)
49   zhat = map(z_all, function(z1) return z1 - z.mu + x.mu end)
50   bigger = 0
51   for j=1,the.boot do
52     if obs( adds(many(yhat,#yhat)),  adds(many(zhat,#zhat))) > b4
53     then bigger = bigger + 1/the.boot end end
54   return bigger >= the.conf end
55
56 -- r ==> rx
57 function scottKnot(nums,      all,cohen,summary,div)
58   function summary(i,j,      out)
59     out = copy(nums[i])
60     for k = i+1, j do out = out:merge(nums[k]) end
61     return out
62   end
63   function div(lo,hi,rank,b4,      cut,best,l1,l,r,r1,now)
64     best = 0
65     for j = lo,hi do
66       if j < hi then
67         l = summary(lo, j)
68         r = summary(j+1, hi)
69         now = (l.n*((l.mu - b4.mu)^2 + r.n*(r.mu - b4.mu)^2)/(l.n+r.n)
70         if now > best then
71           if math.abs(l.mu - r.mu) >= cohen then
72             cut, best, l1, r1 = j, now, copy(l), copy(r)
73           end end end
74         if cut and not l1:same(r1,the) then
75           rank = div(lo,      cut, rank, l1) + 1
76           rank = div(cut+1, hi, rank, r1)
77         else
78           for i = lo,hi do nums[i].rank = rank end end
79       return rank
80     end
81     table.sort(nums, function(x,y) return mid(x) < mid(y) end)
82     all = summary(1,#nums)
83     cohen = all.sd * the.cohen
84     div(1, #nums, 1, all)
85     return nums end
86
87
88 function demo( normal,stats, mu)
89   function normal(mu,sd)
90     return mu + sd*math.sqrt(-2*math.log(R()))*math.cos(2*math.pi*R()) end
91   stats = {a={},b={},c={},d={},e={}}
92   mu=5
93   for _,t in pairs(stats) do
94     for i=1,100 do t[i+#t]=normal(mu,3) end; mu = mu + 5 end
95   for _,x in pairs(quintiles(stats)) do
96     print(table.concat(x.pers,""),x.display) end
97

```