

```

1  -----
2  vim: ts=2 sw=2 et :
3  local b4_help = {}
4  LESSISMORE: best or test multi-objective optimization.
5  (c) 2022 Tim Menzies, tim@ieee.org
6  "I think the highest and lowest points are the important ones.
7  Anything else is just...in between." - Jim Morrison
8
9  USAGE:
10 alias lim="lua lessismore.lua"
11 lim [OPTIONS]
12
13 OPTIONS:
14 -H --how good or bad or novel = good
15 -m --min exponent of min size = .5
16 -b --bins max bins = 16
17 --seed random number seed = 10019
18 -S --some number of nums to keep = 256
19 -p --ppm exponent of distance = 2
20
21
22 OPTIONS (other):
23 -f --file where to find data = ../etc/data/auto93.csv
24 -h --help show help
25 -r --rnd rounding rules = %5.2f
26 -g --go start up action = nothing
27
28 Usage of the works is permitted provided that this instrument is
29 assigned with the works, so that any entity that uses the works is
30 notified of this instrument.  DISCLAIMER:THE WORKS ARE WITHOUT WARRANTY.  ]
31
32 -- ## Namespace
33 local the={}
34 local big,copy,csv,demos,discretize,dist,eq,entropy,fill_in_the,fmt,gap,is,like,lt
35 local mac,merge,mid,mode,mu,nasa93dem,norm,num,o,oo,pdf,per,push,rand,range
36 local rnd,rnds,row84,slice,sort,some,same,sd,string2thing,sym
37 local NUM,SYM,RANGE,EGS,COLS,ROW
38 for k,v in pairs(_ENV) do b4[k]=k end -- At end, use 'b4' to find rogue vars.
39
40 -- ## Coding Conventions
41 -- Separate policy from mechanism.
42 -- For commonly revised parts of this processing (e.g. the name and type of data
43 -- columns) define a little language to support easy revision.
44 -- Also, all 'magic parameters' that control code behavior should be part
45 -- of the help text. Parse that string to set those options.
46 -- Allow for '-h' on the command line to print that help. Allow other command
47 -- line flags to update those options.
48 -- Dialogue independence.
49 -- Isolate and separate operating system interaction.
50 -- Test-driven development.
51 -- The 'go' functions store tests.
52 -- Tests should be silent unless they -- fail. -tests can be
53 -- disabled by renaming from 'go.fun' to 'no.fun'. Tests should
54 -- return 'true' if the test passes. On exit, return number of
55 -- failed tests.
56 -- Write less code.
57 -- "One of my most productive days was throwing away 1,000 lines of code."
58 -- (Ken Thompson);
59 -- "It is vain to do with more what can be done with less."
60 -- (William of Occam);
61 -- "less, but better"
62 -- (Dieter Rams);
63 -- Good code is short code. If you know what is going on, the code
64 -- is shorter. While the code is longer, find patterns of processing
65 -- that combines N things into less things. Strive to write shorter.
66 -- Lots of short functions. Methods listed alphabetically.
67 -- Code 80 chars wide, or less. Functions in 1 line.
68 -- If you can. Indent with two spaces. Divide code into 120 line (or
69 -- less) pages. Use 'i' instead of 'self'.
70 -- Minimize use of local (exception: define all functions
71 -- local at top of file).
72 -- Encapsulation:
73 -- Use polymorphism but no inheritance (simpler
74 -- debugging). All classes get a 'new' constructor.
75 -- Use UPPER CASE for class names.
76 -- Class,Responsibilities,Collaborators.
77 -- Each class is succinctly documented as a set of collaborations
78 -- to fulfill some responsibility.
79 -- Falsifiable.
80 -- Code does something. It should be possible to say when that thing
81 -- is not happening. See external and internal metrics (Fenton).
82
83 -- ## About the Learning
84 -- Data is stored in ROWs.
85 -- Beware missing values (marked in "?") and avoid them
86 -- Where possible all learning should be incremental.
87 -- Standard deviation and entropy generalized to 'div' (diversity);
88 -- Mean and mode generalized to 'mid' (middle);
89 -- Rows are created once and shared between different sets of
90 -- examples (so we can accumulate statistics on how we are progressing
91 -- inside each row)
92 -- When a row is first created, it is assigned to a 'base', i.e.
93 -- a place to store the 'lo,hi' values for all numerics.
94 -- XXX tables very useful
95 -- XXX table have cols. cols are num, syms. ranges
96

```

```

97 function nasa93dem()
98 local vl,l,n,h,vh,xhsl,2,3,4,5,6; return {
99   ["id","center","Year","prec","flex","resl","team","pmat","rely","data","cpix",
100    "ruse","docu","time","stor","pvol","acap","pcap","pcon",
101    "apex","plex","flex","tool","site","sced","kloc",
102    "effort","Defects","Months"],
103   {
104     (1,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,l,n,n,n,n,h,n,l,25,9,117,6,808,15,3),
105     (2,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,24,6,117,6,767,15),
106     (3,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,7,7,31,2,240,10,1),
107     (4,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,8,2,36,256,10,4),
108     (5,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,9,7,25,2,302,11),
109     (6,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,2,2,8,4,69,6),
110     (7,2,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,3,5,10,8,109,7,8),
111     (8,2,1982,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,66,6,352,8,2077,21),
112     (9,1,1980,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,7,5,72,226,13,6),
113     (10,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,vh,n,vh,n,n,n,20,72,566,14,4),
114     (11,1,1984,h,h,h,vh,n,n,l,h,n,n,n,n,l,h,h,n,vh,n,h,n,n,6,24,188,9,9),
115     (12,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,vh,n,vh,n,h,n,n,100,360,2832,25,2),
116     (13,1,1985,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,n,vh,n,l,n,n,n,11,3,36,456,12,8),
117     (14,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,n,n,n,n,31,5,60,986,17,6),
118     (15,1,1983,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,h,n,vh,n,h,n,n,20,48,626,15,1),
119     (16,1,1982,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,n,n,n,n,v,l,n,n,n,100,360,4342,28),
120     (17,1,1980,h,h,h,vh,n,n,l,h,n,n,n,n,xh,l,h,vh,n,vh,n,h,n,n,150,324,4868,32,5),
121     (18,1,1984,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,n,n,n,h,n,n,31,5,60,986,17,6),
122     (19,1,1983,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,h,n,n,vh,n,h,n,n,15,48,470,13,6),
123     (20,1,1984,h,h,h,vh,n,n,l,h,n,n,n,n,xh,l,h,n,n,n,h,n,n,n,32,5,60,1276,20,8),
124     (21,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,13,7,60,614,13,9),
125     (22,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,66,6,352,8,2077,21),
126     (23,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,29,5,120,920,16),
127     (24,2,1986,h,h,h,vh,n,n,h,n,n,n,n,n,n,n,h,n,h,n,n,n,n,15,90,575,15,2),
128     (25,2,1986,h,h,h,vh,n,n,h,n,n,n,n,n,n,n,h,n,h,n,n,n,n,38,210,1553,21,3),
129     (26,2,1986,h,h,h,vh,n,n,h,n,n,n,n,n,n,n,h,n,h,n,n,n,n,10,48,1268,9820,37,3),
130     (27,2,1982,h,h,h,vh,n,n,vh,n,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,15,4,70,765,14,5),
131     (28,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,h,n,l,vh,n,n,h,l,h,n,n,1,48,5,239,2409,21,4),
132     (29,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,1,6,3,82,810,14,8),
133     (30,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,h,n,h,l,vh,n,n,h,l,h,n,n,1,12,8,168,9820,37,3),
134     (31,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,vh,l,vh,n,n,h,l,h,n,n,1,32,6,170,619,18,7),
135     (32,2,1982,h,h,h,vh,h,n,vh,h,n,n,vh,h,n,l,vh,n,n,h,l,h,n,n,1,35,5,192,1763,19,3),
136     (33,2,1985,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,5,5,18,172,9,1),
137     (34,2,1987,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,10,4,30,324,11,2),
138     (35,2,1987,h,h,h,vh,n,n,n,n,n,n,n,n,n,n,h,n,n,n,n,14,60,457,12,4),
139     (36,2,1986,h,h,h,vh,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,6,5,42,290,12),
140     (37,2,1986,h,h,h,vh,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,13,60,683,14,4),
141     (38,2,1986,h,h,h,vh,h,n,n,n,n,n,n,n,n,n,n,h,n,n,n,n,90,444,334,26,7),
142     (39,2,1986,h,h,h,vh,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,8,45,420,42,3),
143     (40,2,1986,h,h,h,vh,n,n,n,n,n,n,h,n,n,n,n,n,n,n,n,16,114,887,16,4),
144     (41,2,1980,h,h,h,vh,h,n,h,n,h,n,n,vh,h,l,h,n,n,n,l,h,n,n,1,177,9,1248,7998,31,5),
145     (42,6,1975,h,h,h,vh,h,h,l,h,n,n,n,n,n,l,n,h,n,n,n,n,302,2400,8543,38,4),
146     (43,5,1979,h,h,h,vh,h,h,l,h,n,n,n,n,n,n,h,n,n,n,n,n,190,420,4907,24,3),
147     (44,5,1982,h,h,h,vh,h,h,l,h,n,n,n,n,n,n,h,n,n,n,n,n,284,7,973,8518,38,1),
148     (45,5,1982,h,h,h,vh,n,n,h,n,n,n,n,n,n,n,l,n,h,n,h,n,n,n,79,400,2327,26,9),
149     (46,5,1977,h,h,h,vh,l,1,n,n,n,n,n,n,n,l,h,vh,n,h,n,h,n,n,423,2400,18447,41,9),
150     (47,5,1977,h,h,h,vh,n,n,h,n,n,n,n,n,n,n,h,vh,n,n,n,n,177,9,1248,7998,31,5),
151     (48,5,1984,h,h,h,vh,n,n,n,n,n,n,n,n,n,n,h,n,h,n,n,n,47,5,252,2007,22,3),
152     (49,5,1980,h,h,h,vh,l,vh,n,xh,n,n,h,h,l,n,n,n,n,h,n,n,21,107,1058,21,3),
153     (50,5,1983,h,h,h,vh,n,n,h,n,n,n,vh,n,n,h,n,h,n,h,n,n,n,78,571,4,4815,30,5),
154     (51,5,1979,h,h,h,vh,l,h,n,h,n,n,n,n,h,n,n,n,n,n,n,4,98,9,701,15,5),
155     (52,5,1985,h,h,h,vh,l,n,n,n,n,n,n,vh,n,n,h,n,h,n,n,n,n,19,3,155,1191,18,6),
156     (53,5,1979,h,h,h,vh,l,h,n,n,vh,n,n,h,h,l,h,n,n,n,n,h,n,n,101,750,4840,32,4),
157     (54,5,1979,h,h,h,vh,l,h,n,n,n,n,h,h,l,n,n,n,n,n,n,n,219,2120,11761,42,8),
158     (55,5,1979,h,h,h,vh,l,h,n,n,n,n,n,h,n,n,n,n,n,n,n,50,370,2685,25,6),
159     (56,2,1979,h,h,h,vh,h,h,h,n,n,n,n,vh,vh,n,vh,vh,n,vh,h,n,l,227,1181,6293,33),
160     (57,2,1977,h,h,h,vh,h,n,h,n,vh,n,n,n,n,l,h,vh,n,n,l,n,n,n,1,70,278,2950,20,2),
161     (58,2,1979,h,h,h,vh,h,h,h,l,h,n,n,n,n,n,l,n,n,n,n,h,n,n,1,0,9,8,4,28,4,9),
162     (59,6,1974,h,h,h,vh,l,vh,l,xh,n,n,xh,vh,l,h,h,n,n,vh,vl,h,n,n,n,980,4560,50961,96),
163     (60,6,1975,h,h,h,vh,n,n,l,h,n,n,n,n,n,l,vh,vh,n,n,h,n,n,n,n,350,720,8547,35,7),
164     (61,5,1976,h,h,h,vh,h,h,n,xh,n,n,h,h,l,h,n,n,n,n,h,h,n,n,70,458,2404,27,5),
165     (62,5,1979,h,h,h,vh,h,h,n,xh,n,n,h,h,l,h,n,n,n,n,h,h,n,n,271,2460,9308,43,4),
166     (63,5,1971,h,h,h,vh,n,n,n,n,n,n,n,n,n,l,h,h,n,h,n,n,n,n,90,162,2743,25),
167     (64,5,1980,h,h,h,vh,n,n,n,n,n,n,n,n,n,l,h,n,n,h,n,n,n,n,40,150,1219,18,9),
168     (65,5,1979,h,h,h,vh,n,n,h,n,n,n,n,h,n,l,h,h,n,h,n,n,n,n,137,636,4210,32,2),
169     (66,5,1977,h,h,h,vh,n,n,h,n,n,n,n,n,n,h,h,l,h,n,n,h,n,n,n,150,882,5848,36,2),
170     (67,5,1976,h,h,h,vh,n,vh,n,h,n,n,h,n,l,h,h,n,h,n,n,n,339,444,8477,45,9),
171     (68,5,1983,h,h,h,vh,n,l,h,l,n,n,n,n,n,h,h,h,n,h,n,n,240,192,10313,37,1),
172     (69,5,1978,h,h,h,vh,l,h,n,n,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,144,576,6129,28,8),
173     (70,5,1979,h,h,h,vh,l,n,l,n,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,151,432,6136,26,2),
174     (71,5,1979,h,h,h,vh,l,n,l,n,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,34,72,1555,16,2),
175     (72,5,1979,h,h,h,vh,l,n,n,n,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,98,300,4907,24,4),
176     (73,5,1979,h,h,h,vh,l,n,n,n,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,85,300,4256,23,2),
177     (74,5,1982,h,h,h,vh,l,n,l,n,n,n,n,n,vh,l,h,n,n,h,h,h,h,n,n,1,20,240,813,12,8),
178     (75,5,1978,h,h,h,vh,l,n,l,n,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,111,600,4511,23,5),
179     (76,5,1978,h,h,h,vh,l,h,vh,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,162,756,7553,32,4),
180     (77,5,1978,h,h,h,vh,l,h,h,vh,n,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,352,1200,17597,42,9),
181     (78,5,1979,h,h,h,vh,l,h,n,n,vh,n,n,n,vh,l,h,h,n,h,h,h,h,n,n,1,165,97,7867,31,5),
182     (79,5,1984,h,h,h,vh,h,h,n,vh,n,n,n,h,h,l,h,n,n,n,h,h,n,n,60,409,2004,24,9),
183     (80,5,1984,h,h,h,vh,h,h,n,vh,n,n,n,h,h,l,h,n,n,n,h,h,n,n,100,703,3340,29,6),
184     (81,2,1980,h,h,h,vh,n,n,h,n,n,n,n,n,n,n,h,n,n,n,n,32,1350,2984,33,6),
185     (82,2,1980,h,h,h,vh,h,h,h,n,n,n,n,vh,xh,h,h,n,h,n,h,h,n,n,53,480,2227,28,8),
186     (83,3,1977,h,h,h,vh,h,h,l,vh,n,n,n,vh,xh,l,vh,vh,n,vh,vl,vl,h,n,n,41,599,1594,23),
187     (84,3,1977,h,h,h,vh,h,h,l,vh,n,n,n,vh,xh,l,vh,n,n,vh,vl,vh,n,n,24,430,933,19,2),
188     (85,5,1977,h,h,h,vh,h,h,h,n,n,n,n,n,n,h,n,n,n,n,n,165,45,78,2,63,4,7),
189     (86,5,1977,h,h,h,vh,h,vh,h,vh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,65,1772,5,2468,34,5),
190     (87,5,1977,h,h,h,vh,h,vh,h,vh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,70,1645,9,2658,35,4),
191     (88,5,1977,h,h,h,vh,h,vh,h,xh,n,n,xh,xh,n,h,h,n,h,h,h,n,n,50,1924,5,2102,34,2),
192     (89,5,1982,h,h,h,vh,h,vh,h,vl,l,h,n,n,n,vh,l,h,n,n,n,25,64,495,15,6),
193     (90,5,1980,h,h,h,vh,h,vh,n,n,n,n,xh,xh,n,h,h,n,h,h,h,n,n,233,8211,8848,53,1),
194     (91,2,1983,h,h,h,vh,n,h,n,n,n,n,vh,vh,h,n,n,n,n,l,n,n,n,16,3,480,1253,21,5),
195     (92,2,1983,h,h,h,vh,n,h,n,n,n,n,n,vh,n,n,n,n,n,l,n,n,n,6,2,12,477,15,4),
196     (93,2,1983,h,h,h,vh,n,h,n,n,n,n,n,vh,vh,h,n,n,n,n,l,n,n,n,3,38,231,12,1) end

```

```

197 -----
198 -- ## Utils
199 -- Misc
200 big=math.huge
201 rand=math.random
202 fmt=string.format
203 same = function(x) return x end
204
205 -- Sorting
206 function sort(t,f) table.sort(t, f); return t end
207 function lt(x) return function(a,b) return a[x] < b[x] end end
208
209 -- Query and update
210 function map(t,f,u) u={};;for k,v in pairs(t) do u[l+#u]=f(v) end; return u end
211 function push(t,x) t[l+#t]=x; return x end
212 function slice(t,i,j,k) return t[i],t[i+1],t[i+2],t[i+3],t[i+4],t[i+5],t[i+6],t[i+7],t[i+8],t[i+9],t[i+10],t[i+11],t[i+12],t[i+13],t[i+14],t[i+15],t[i+16],t[i+17],t[i+18],t[i+19],t[i+20],t[i+21],t[i+22],t[i+23],t[i+24],t[i+25],t[i+26],t[i+27],t[i+28],t[i+29],t[i+30],t[i+31],t[i+32],t[i+33],t[i+34],t[i+35],t[i+36],t[i+37],t[i+38],t[i+39],t[i+40],t[i+41],t[i+42],t[i+43],t[i+44],t[i+45],t[i+46],t[i+47],t[i+48],t[i+49],t[i+50],t[i+51],t[i+52],t[i+53],t[i+54],t[i+55],t[i+56],t[i+57],t[i+58],t[i+59],t[i+60],t[i+61],t[i+62],t[i+63],t[i+64],t[i+65],t[i+66],t[i+67],t[i+68],t[i+69],t[i+70],t[i+71],t[i+72],t[i+73],t[i+74],t[i+75],t[i+76],t[i+77],t[i+78],t[i+79],t[i+80],t[i+81],t[i+82],t[i+83],t[i+84],t[i+85],t[i+86],t[i+87],t[i+88],t[i+89],t[i+90],t[i+91],t[i+92],t[i+93],t[i+94],t[i+95],t[i+96],t[i+97],t[i+98],t[i+99],t[i+100],t[i+101],t[i+102],t[i+103],t[i+104],t[i+105],t[i+106],t[i+107],t[i+108],t[i+109],t[i+110],t[i+111],t[i+112],t[i+113],t[i+114],t[i+115],t[i+116],t[i+117],t[i+118],t[i+119],t[i+120],t[i+121],t[i+122],t[i+123],t[i+124],t[i+125],t[i+126],t[i+127],t[i+128],t[i+129],t[i+130],t[i+131],t[i+132],t[i+133],t[i+134],t[i+135],t[i+136],t[i+137],t[i+138],t[i+139],t[i+140],t[i+141],t[i+142],t[i+143],t[i+144],t[i+145],t[i+146],t[i+147],t[i+148],t[i+149],t[i+150],t[i+151],t[i+152],t[i+153],t[i+154],t[i+155],t[i+156],t[i+157],t[i+158],t[i+159],t[i+160],t[i+161],t[i+162],t[i+163],t[i+164],t[i+165],t[i+166],t[i+167],t[i+168],t[i+169],t[i+170],t[i+171],t[i+172],t[i+173],t[i+174],t[i+175],t[i+176],t[i+177],t[i+178],t[i+179],t[i+180],t[i+181],t[i+182],t[i+183],t[i+184],t[i+185],t[i+186],t[i+187],t[i+188],t[i+189],t[i+190],t[i+191],t[i+192],t[i+193],t[i+194],t[i+195],t[i+196],t[i+197],t[i+198],t[i+199],t[i+200],t[i+201],t[i+202],t[i+203],t[i+204],t[i+205],t[i+206],t[i+207],t[i+208],t[i+209],t[i+210],t[i+211],t[i+212],t[i+213],t[i+214],t[i+215],t[i+216],t[i+217],t[i+218],t[i+219],t[i+220],t[i+221],t[i+222],t[i+223],t[i+224],t[i+225],t[i+226],t[i+227],t[i+228],t[i+229],t[i+230],t[i+231],t[i+232],t[i+233],t[i+234],t[i+235],t[i+236],t[i+237],t[i+238],t[i+239],t[i+240],t[i+241],t[i+242],t[i+243],t[i+244],t[i+245],t[i+246],t[i+247],t[i+248],t[i+249],t[i+250],t[i+251],t[i+252],t[i+253],t[i+254],t[i+255],t[i+256],t[i+257],t[i+258],t[i+259],t[i+260],t[i+261],t[i+262],t[i+263],t[i+264],t[i+265],t[i+266],t[i+267],t[i+268],t[i+269],t[i+270],t[i+271],t[i+272],t[i+273],t[i+274],t[i+275],t[i+276],t[i+277],t[i+278],t[i+279],t[i+280],t[i+281],t[i+282],t[i+283],t[i+284],t[i+285],t[i+286],t[i+287],t[i+288],t[i+289],t[i+290],t[i+291],t[i+292],t[i+293],t[i+294],t[i+295],t[i+296],t[i+297],t[i+298],t[i+299],t[i+300],t[i+301],t[i+302],t[i+303],t[i+304],t[i+305],t[i+306],t[i+307],t[i+308],t[i+309],t[i+310],t[i+311],t[i+312],t[i+313],t[i+314],t[i+315],t[i+316],t[i+317],t[i+318],t[i+319],t[i+320],t[i+321],t[i+322],t[i+323],t[i+324],t[i+325],t[i+326],t[i+327],t[i+328],t[i+329],t[i+330],t[i+331],t[i+332],t[i+333],t[i+334],t[i+335],t[i+336],t[i+337],t[i+338],t[i+339],t[i+340],t[i+341],t[i+342],t[i+343],t[i+344],t[i+345],t[i+346],t[i+347],t[i+348],t[i+349],t[i+350],t[i+351],t[i+352],t[i+353],t[i+354],t[i+355],t[i+356],t[i+357],t[i+358],t[i+359],t[i+360],t[i+361],t[i+362],t[i+363],t[i+364],t[i+365],t[i+366],t[i+367],t[i+368],t[i+369],t[i+370],t[i+371],t[i+372],t[i+373],t[i+374],t[i+375],t[i+376],t[i+377],t[i+378],t[i+379],t[i+380],t[i+381],t[i+382],t[i+383],t[i+384],t[i+385],t[i+386],t[i+387],t[i+388],t[i+389],t[i+390],t[i+391],t[i+392],t[i+393],t[i+394],t[i+395],t[i+396],t[i+397],t[i+398],t[i+399],t[i+400],t[i+401],t[i+402],t[i+403],t[i+404],t[i+405],t[i+406],t[i+407],t[i+408],t[i+409],t[i+410],t[i+411],t[i+412],t[i+413],t[i+414],t[i+415],t[i+416],t[i+417],t[i+418],t[i+419],t[i+420],t[i+421],t[i+422],t[i+423],t[i+424],t[i+425],t[i+426],t[i+427],t[i+428],t[i+429],t[i+430],t[i+431],t[i+432],t[i+433],t[i+434],t[i+435],t[i+436],t[i+437],t[i+438],t[i+439],t[i+440],t[i+441],t[i+442],t[i+443],t[i+444],t[i+445],t[i+446],t[i+447],t[i+448],t[i+449],t[i+450],t[i+451],t[i+452],t[i+453],t[i+454],t[i+455],t[i+456],t[i+457],t[i+458],t[i+459],t[i+460],t[i+461],t[i+462],t[i+463],t[i+464],t[i+465],t[i+466],t[i+467],t[i+468],t[i+469],t[i+470],t[i+471],t[i+472],t[i+473],t[i+474],t[i+475],t[i+476],t[i+477],t[i+478],t[i+479],t[i+480],t[i+481],t[i+482],t[i+483],t[i+484],t[i+485],t[i+486],t[i+487],t[i+488],t[i+489],t[i+490],t[i+491],t[i+492],t[i+493],t[i+494],t[i+495],t[i+496],t[i+497],t[i+498],t[i+499],t[i+500],t[i+501],t[i+502],t[i+503],t[i+504],t[i+505],t[i+506],t[i+507],t[i+508],t[i+509],t[i+510],t[i+511],t[i+512],t[i+513],t[i+514],t[i+515],t[i+516],t[i+517],t[i+518],t[i+519],t[i+520],t[i+521],t[i+522],t[i+523],t[i+524],t[i+525],t[i+526],t[i+527],t[i+528],t[i+529],t[i+530],t[i+531],t[i+532],t[i+533],t[i+534],t[i+535],t[i+536],t[i+537],t[i+538],t[i+539],t[i+540],t[i+541],t[i+542],t[i+543],t[i+544],t[i+545],t[i+546],t[i+547],t[i+548],t[i+549],t[i+550],t[i+551],t[i+552],t[i+553],t[i+554],t[i+555],t[i+556],t[i+557],t[i+558],t[i+559],t[i+560],t[i+561],t[i+562],t[i+563],t[i+564],t[i+565],t[i+566],t[i+567],t[i+568],t[i+569],t[i+570],t[i+571],t[i+572],t[i+573],t[i+574],t[i+575],t[i+576],t[i+577],t[i+578],t[i+579],t[i+580],t[i+581],t[i+582],t[i+583],t[i+584],t[i+585],t[i+586],t[i+587],t[i+588],t[i+589],t[i+590],t[i+591],t[i+592],t[i+593],t[i+594],t[i+595],t[i+596],t[i+597],t[i+598],t[i+599],t[i+600],t[i+601],t[i+602],t[i+603],t[i+604],t[i+605],t[i+606],t[i+607],t[i+608],t[i+609],t[i+610],t[i+611],t[i+612],t[i+613],t[i+614],t[i+615],t[i+616],t[i+617],t[i+618],t[i+619],t[i+620],t[i+621],t[i+622],t[i+623],t[i+624],t[i+625],t[i+626],t[i+627],t[i+628],t[i+629],t[i+630],t[i+631],t[i+632],t[i+633],t[i+634],t[i+635],t[i+636],t[i+637],t[i+638],t[i+639],t[i+640],t[i+641],t[i+642],t[i+643],t[i+644],t[i+645],t[i+646],t[i+647],t[i+648],t[i+649],t[i+650],t[i+651],t[i+652],t[i+653],t[i+654],t[i+655],t[i+656],t[i+657],t[i+658],t[i+659],t[i+660],t[i+661],t[i+662],t[i+663],t[i+664],t[i+665],t[i+666],t[i+667],t[i+668],t[i+669],t[i+670],t[i+671],t[i+672],t[i+673],t[i+674],t[i+675],t[i+676],t[i+677],t[i+678],t[i+679],t[i+680],t[i+681],t[i+682],t[i+683],t[i+684],t[i+685],t[i+686],t[i+687],t[i+688],t[i+689],t[i+690],t[i+691],t[i+692],t[i+693],t[i+694],t[i+695],t[i+696],t[i+697],t[i+698],t[i+699],t[i+700],t[i+701],t[i+702],t[i+703],t[i+704],t[i+705],t[i+706],t[i+707],t[i+708],t[i+709],t[i+710],t[i+711],t[i+712],t[i+713],t[i+714],t[i+715],t[i+716],t[i+717],t[i+718],t[i+719],t[i+720],t[i+721],t[i+722],t[i+723],t[i+724],t[i+725],t[i+726],t[i+727],t[i+728],t[i+729],t[i+730],t[i+731],t[i+732],t[i+733],t[i+734],t[i+735],t[i+736],t[i+737],t[i+738],t[i+739],t[i+740],t[i+741],t[i+742],t[i+743],t[i+744],t[i+745],t[i+746],t[i+747],t[i+748],t[i+749],t[i+750],t[i+751],t[i+752],t[i+753],t[i+754],t[i+755],t[i+756],t[i+757],t[i+758],t[i+759],t[i+760],t[i+761],t[i+762],t[i+763],t[i+764],t[i+765],t[i+766],t[i+767],t[i+768],t[i+769],t[i+770],t[i+771],t[i+772],t[i+773],t[i+774],t[i+775],t[i+776],t[i+777],t[i+778],t[i+779],t[i+780],t[i+781
```

```

273 -----
274 -- ## Objects
275 COLS,EGS,NUM,RANGE,ROW,SYM=is"COLS",is"EGS",is"NUM",is"RANGE",is"SYM",is"ROW"
276 -----
277 -- ## NUM
278 -- For a stream of 'add'itions, incrementally maintain 'mu,sd'.
279 -- - 'Norm'alize data for distance and discretization calcs
280 -- (see 'dist' and 'range').
281 -- - Comment on 'like'lihood that something belongs to this distribution.
282 function NUM.new(i,at,txt)
283   i.at=at or 0; i.txt=txt or ""; i.lo,i.hi=big, -big
284   i.n,i.mu,i.m2,i.sd = 0,0,0,0; i.w=(txt or ""):find"--$* and -1 or 1 end
285
286 function NUM.add(i,x, d)
287   if x=="?" then return x end
288   i.n = i.n + 1
289   d = x - i.mu
290   i.mu = i.mu + d/i.n
291   i.m2 = i.m2 + d*(x - i.mu)
292   i.sd = (i.m2/0.000001 + i.n^2) and 0 or ((i.m2/(i.n - 1))^0.5)
293   i.lo = math.min(i.lo,x)
294   i.hi = math.max(i.hi,x) end
295
296 function NUM.dist(i,x,y)
297   if x=="?" and y=="?" then return 1 end
298   if x=="?" then y = i:norm(y); x = y<.5 and 1 or 0
299   elseif y=="?" then x = i:norm(x); y = x<.5 and 1 or 0
300   else x,y = i:norm(x), i:norm(y) end
301   return math.abs(x - y) end
302
303 function NUM.like(i,x,_, e)
304   return (x < i.mu - 4*i.sd and 0 or x > i.mu + 4*i.sd and 0 or
305     2.7183*(-(x - i.mu)^2 / (x + 2*i.sd^2))/(z + (math.pi^2*i.sd^2)^.5)) end
306
307 function NUM.merge(i,ranges,min, a,b,c,j,n,tmp)
308   function expand(t)
309     if t<2 then return {} end
310     for j=2,#t do t[j].lo=t[j-1].hi end
311     t[1].x.lo, t[#t].x.hi = -big,big
312     return t
313   end
314   j,n,tmp = 1,ranges,{}
315   while j<#n do
316     a, b = ranges[j], ranges[j+1]
317     if b then c = a:merge(b,min); if c then a,j = c,j+1 end end
318     tmp[#tmp+1] = a
319     j = j+1 end
320   return #tmp==#ranges and expand(tmp) or i:merge(tmp,min) end
321
322 function NUM.mid(i) return i.mu end
323
324 function NUM.norm(i,x)
325   return i.hi-i.lo<1E-9 and 0 or (x-i.lo)/(i.hi-i.lo+1/big) end
326
327 function NUM.range(i,x,n, b) b=(i.hi-i.lo)/n; return math.floor(x/b+0.5)*b end
328 -----
329 -- ## SYM
330 -- For a stream of 'add'itions, incrementally maintain count of 'all' symbols.
331 -- - Using that info, report 'dist', mode ('mid') symbol, and entropy
332 -- ('div') of this distribution.
333 -- - Comment on 'like'lihood that something belongs to this distribution.
334 -- - Discretization of a symbol just returns that sym ('range').
335 function SYM.new(i,at,txt) i.at=at or 0; i.txt=txt or ""; i.n,i.all = 0,{} end
336
337 function SYM.add(i,x,n)
338   if x=="?" then return x end
339   n = n or 1
340   i.n=i.n+n; i.all[x] = n + (i.all[x] or 0) end
341
342 function SYM.dist(i,x,y) return (a==b and 0 or 1) end
343
344 function SYM.div(i, n,e)
345   e=0; for k,n in pairs(i.all) do e=e-n/i.n*math.log(n/i.n,2) end ;return e end
346
347 function SYM.like(i,x,prior) return ((c.all[x] or 0)+the.m*prior)/(c.n+the.m) end
348
349 function SYM.merge(i,ranges,min) return ranges end
350
351 function SYM.mid(i)
352   m=0; for y,n in pairs(i.all) do if n>m then m,x=n,y end end; return x end
353
354 function SYM.range(i,x,_) return x end

```

```

355 -----
356 -- ## RANGE
357 -- - For a stream of 'add'itions, incrementally maintain counts of 'x' and 'y'.
358 -- - Summarize 'x' as the 'lo,hi' seen so far and summarize 'y' in 'SYM' counts
359 -- in 'y.all' (and get counts there using 'of').
360 -- - Support range sorting ('_lt') and printing ('__tostring').
361 -- - Check if this range's 'x' values 'select's for a particular row.
362 -- - Merge 'adjacent' ranges if the entropy of the whole is less than the parts.
363 function RANGE.new(i,col,lo,hi,y)
364   i.col, i.x, i.y = col, (lo==0 or big, hi==hi or -big), (y or SYM()) end
365
366 function RANGE.__lt(i,j) return i.x.lo < j.x.lo end
367
368 function RANGE.__tostring(i)
369   local x, lo, hi = i.col.txt, i.x.lo, i.x.hi
370   if lo == hi then return fmt("%s==%s",x, lo)
371   elseif hi == big then return fmt("%s>=%s",x, lo)
372   elseif lo == -big then return fmt("%s<=%s", x, hi)
373   else return fmt("%s<=%s<%s",lo,x,hi) end end
374
375 function RANGE.add(i,x,y)
376   if x=="?" then return x end
377   i.x.lo = math.min(i.x.lo,x)
378   i.x.hi = math.max(i.x.hi,x)
379   i.y:add(y) end
380
381 function RANGE.merge(i,j,n0, k)
382   k = SYM(i.col.at, i.col.txt)
383   for x,n in pairs(i.y.all) do k:add(x,n) end
384   for x,n in pairs(j.y.all) do k:add(x,n) end
385   if i.y.n<(n0 or 0) or j.y.n<(n0 or 0) or (
386     (i.y:div(i)*i.y.n + j.y:div(j)*j.y.n)/k.n >= .99*k:div())
387   then return RANGE(i.col, i.x.lo, j.x.hi, k) end end
388
389 function RANGE.of(i,x) return i.y.all[x] or 0 end
390
391 function RANGE.score(i,goal,B,R, how)
392   how={}
393   how.good= function(b,r) return ((b<r or b+r < .05) and 0) or b^2/(b+r) end
394   how.bad= function(b,r) return ((r<b or b+r < .05) and 0) or r^2/(b+r) end
395   how.noel=function(b,r) return 1/(b+r) end
396   b, r, z = 0, 0, 1/big
397   for x,n in pairs(i.y.all) do
398     if x==goal then b = b+n else r=r+n end end
399   return how[the.how or "good"] (b/(B+z), z/(R+z)) end
400
401 function RANGE.selects(i,t, x)
402   t = t.cells and t.cells or t
403   x = t[i.at]
404   return x=="?" or (i.x.lo==i.x.hi and i.x.lo==x) or (i.x.lo<x and x<i.x.hi) end
405 -----
406 -- ## ROW
407 -- - Using knowledge 'of' the geometry of the data, support distance calcs
408 -- i ('_sub' and '_around') as well as multi-objective ranking ('__lt').
409 function ROW.new(i,eg, cells) i.of,i.cells = eg,cells end
410
411 function ROW.__lt(i,j, s1,s2,e,y,a,b)
412   y = i.of.cols.y
413   s1, s2, e = 0, 0, math.exp(1)
414   for _,col in pairs(y) do
415     a = col:norm(i.cells[col.at])
416     b = col:norm(j.cells[col.at])
417     s1 = s1 - e^(col.w * (a - b) / #y)
418     s2 = s2 - e^(col.w * (b - a) / #y) end
419   return s1/#y < s2/#y end
420
421 function ROW.__sub(i,j)
422   for _,col in pairs(i.of.cols.x) do
423     a,b = i.cells[col.at], j.cells[col.at]
424     inc = a=="?" and b=="?" and 1 or col:dist(a,b)
425     d = d + inc*the.p end
426   return (d / (#i.of.cols.x)) ^ (1/the.p) end
427
428 function ROW.around(i,rows)
429   return sort(map(rows or i.of.rows, function(j) return (dist=i-j,row=j) end),
430     i.txt") end
431

```

```

431 -----
432 -- ## COLS
433 -- - Factory for converting column 'names' to 'NUM's ad 'SYM's.
434 -- - Store all columns in -- 'all', and for all columns we are not skipping,
435 -- store the independent and dependent columns distributions in 'x' and 'y'.
436 function COLS.new(i,names, head,row,col)
437   i.names=names; i.all={}; i.y={}; i.x={}
438   for at,txt in pairs(names) do
439     col = push(i.all, (txt:find"*[A-Z]" and NUM or SYM) (at, txt))
440     col.goalp = txt:find"[+]=S" and true or false
441     if not txt:find"$" then
442       if txt:find"$" then i.klass=col end
443       push(col.goalp and i.y or i.x, col) end end end
444 -----
445 -- ## EGS
446 -- - For a stream of 'add'itions, incrementally store rows, summarized in 'cols'.
447 -- - When 'adding', build new rows for new data. Otherwise reuse rows across
448 -- multiple sets of examples.
449 -- - Supporting 'copy'ing of this structure, without or without rows of data.
450 -- - Replaces how much this set of examples 'like' a new row.
451 -- - Discretize columns as 'ranges' that distinguish two sets of rows
452 -- (merging irrelevant distinctions).
453 -- - Summarize the 'mid'point of these examples.
454 function EGS.new(i,names) i.rows,i.cols = {}, COLS(names) end
455
456 function EGS.add(i,row, cells)
457   cells = push(i.rows, row.cells and row or ROW(i,row)).cells
458   for n,col in pairs(i.cols.all) do col:add(cells[n]) end end
459
460 function EGS.copy(i,rows, j)
461   j=EGS(i.cols.names); for _,r in pairs(rows or {}) do j:add(r) end;return j end
462
463 function EGS.like(i,t,overall, nHypotheses, c)
464   prior = (#i.rows + the.k) / (overall + the.k * nHypotheses)
465   like = math.log(prior)
466   for at,x in pairs(t) do
467     col=i.cols.all[at]
468     if x=="?" and not c.goalp then
469       like = math.log(col:like(x)) + like end end
470   return like end
471
472 function EGS.load(src, i)
473   if src==nil or type(src)=="string"
474   then for row in csv(src) do if i then i:add(row) else i=EGS(row)end end
475   else for _,row in pairs(src) do if i then i:add(row) else i=EGS(row)end end end
476   return i end
477
478 function EGS.mid(i,cols)
479   return map(cols or i.cols.y, function(c) return c:mid(i) end) end
480
481 function EGS.ranges(i,yes,no, out,x,bin,tmp,score)
482   out={}
483   for _,col in pairs(i.cols.x) do -- for each x col
484     tmp = {} -- find ranges that distinguish yes and no
485     for _,what in pairs((rows=yes, klass=true), (rows=no, klass=false)) do
486       for _,row in pairs(what.rows) do x = row.cells[col.at]
487         if x=="?" then
488           bin = col:range(x,the.bins)
489           tmp[bin] = tmp[bin] or RANGE(col,x,x)
490           tmp[bin]:add(x, what.klass) end end end
491     tmp = map(tmp,same) -- a hack. makes tmp sortable (has consecutive indexes)
492     for _,range in pairs(col:merge(sort(tmp), (#yes+no)*the.min)) do
493       push(out,range) end end
494   score = function(range) return range:score(true, #yes, #no) end
495   return sort(out,score) end

```

```

496 -----
497 -- ## Code for tests and demos
498
499 -- Simple stuff
500 function go.the()      return type(the.bins)=="number" end
501 function go.sort( t)  return 0==sort((100,3,4,2,10,0))[1] end
502 function go.slice( t,u)
503   t = {10,20,30,40,50,60,70,80,90,100,110,120,130,140}
504   u = slice(t,3,#t,3)
505   t = slice(t,3,5)
506   return #t==3 and #u==4 end
507
508 function go.num(      n,mu,sd)
509   n, mu, sd = NUM(), 10, 1
510   for i=1,10^4 do
511     n=add(mu*sd*math.sqrt(-2*math.log(rand()))*math.cos(2*math.pi*rand())) end
512     return math.abs(n.mu - mu) < 0.05 and math.abs(n.sd - sd) < 0.5 end
513
514 -- Can we read rows off the disk?
515 function go.rows( n,m)
516   m,n=0,0; for row in csv(the.file) do m=m+1; n=n+#row; end; return n/m==8 end
517
518 -- Can we turn a list of names into columns?
519 function go.cols( i)
520   i=COLS("Name","Age","ShoeSize-")
521   return i.y[1].w == -1 end
522
523 -- Can we read data, summarized as columns?
524 function go.egs( it)
525   it=EGS.load(nasa93dem())
526   return math.abs(it.cols.y[1].mu - 624) < 1 end
527   --for _,row in pairs(nasa93dem())do oo(row) end end
528   --it = EGS.load(the.file); return math.abs(2970 - it.cols.y[1].mu) < 1 end
529
530 -- Does discretization work?
531 function go.ranges( it,n,best,rest,min)
532   --it = EGS.load(the.file)
533   print(the.how)
534   it=EGS.load(nasa93dem())
535   print("all",o(rnds(it:mid()))))
536   it.rows = sort(it.rows)
537   for j,row in pairs(sort(it.rows)) do row.klass = 1+j//(#it.rows*.35/6) end
538   n = (#it.rows)^.5
539   best,rest = slice(it.rows,1,n), slice(it.rows, n+1, #it.rows, 3*n)
540   print("best",#best,o(rnds(it:copy(best):mid()))))
541   print("rest",#rest,o(rnds(it:copy(rest):mid()))))
542   tmp={}; for _,ranges in pairs(it:ranges(best,rest)) do
543     for at,range in pairs(ranges) do
544       push(tmp,range).val= range:score(true,#best,#rest) end end
545     for _,range in pairs(sort(tmp,lt*val)) do print(range.val, range) end
546     --oo(a:mid())
547     --oo(b:mid())
548     return math.abs(2970 - it.cols.y[1].mu) < 1 end

```

```

549 -----
550 -- ## Main
551
552 -- - Parse help text for flags and defaults, check CLI for updates.
553 -- - Maybe print the help (with some pretty colors).
554 -- - Run the demos.
555 -- - Check for rogue vars.
556 -- - Exit, reporting number of failures.
557 help:gsub("un ([-|^%s+)([%s]+)([-|~|(|(%s+)|^n)*%s([%s]+)",fill_in_the)
558 if the.help then
559   print(help:gsub("%u%u+", "%127[31m%127[0m")
560         :gsub("([%s]+)([-|~|(|(%s+)([%s]+)%s)%s", "%127[33m%2127[0m%3")", ""))
561 else
562   local fails = demos()
563   for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
564   os.exit(fails) end

```