

```

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
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
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
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040

```

```

90 -- ## NUM
91 function NUM:add(s,n) --- Update
92 if x == "s" then
93     local d = self.mu + 1
94     self.mu = self.mu + d/self.n
95     self.m2 = self.m2 + d*(x-self.mu)
96     self.sd = self.rv0 and 0 or (self.m2*x0 and 0 or (self.m2*(self.n-1))^.5)
97     if x == self.n then self.sd = s end
98     if x < self.i0 then self.i0 = x end end end
99
100 function NUM:like(x,...) --- how much does NUM like "x"?
101 return self.sdo and pdf(x,self.mu,self.sd) or (x==self.mu and 1 or 1/big) end
102
103 function NUM:m() --- central tendency
104 return self.mu end
105
106 function NUM:div() --- spread
107 return self.sd end
108
109 -- ## SYM
110 function SYM:add(s,n) --- Update.
111 if s=="t" then
112     self.s = self.n + 1
113     self.has[s] = 1 + self.has[s] or 0
114     if self.has[s]>nPriorior then
115         self.mostSelf.mode = self.has[s], s end end end
116
117 function SYM:s,>priorior --- how much does SYM like "n"?
118 return (self.has[s]>0) or t.m>nPriorior / (self.n+the.m) end
119
120 function SYM:m() --- central tendency
121 return self.mode end
122
123 function SYM:div(n) --- spread
124 local function fun(p) return p*math.log(p,2) end
125 e=0; for x,y in pairs(self.has) do if n>0 then e = - fun(n/self.n) end end
126 return e end
127
128 -- ## DATA
129 -- ## Create
130 function DATA:clone(src) --- copy structure
131 return add(DATA({self.cols.names[]},src) end
132
133 -- ## Updates
134 function DATA:add(row) --- new row is a header, or a data row
135 if #self.cols.all==0 then self.header=row else self.body=row() end end
136
137 function DATA:header(row) --- Create 'NUM' and 'SYM' for the column headers
138 local col = self.sfindN(1)
139 for n,s in pairs(row) do
140     local col = push(self.cols.all, {sfind("A-Z") and NUM or SYM}(n,s))
141     if sfind"N" then
142         self.cols.klass.col = self.cols.x, col end end end
143     push(self.col["t"] and self.cols.y or self.cols.x, col) end end
144
145 function DATA:body(row) --- Create new row. Store in 'rows'. Update cols.
146 push(self.rows, row)
147 push(self.row.cells and row.col or self.cols.x, self.cols.y) do
148     for _col in pairs(cols) do
149         col=add(row.cells[col.at]) end end end
150
151 function DATA:like(row,n,rows) --- how much DATA likes "row"?
152 local prior,like,int,s
153 prior = (#self.rows + the.k) / (nrows + the.k * nh)
154 like = math.log(prior)
155 row = row.cells and row.cells or row
156 for _col in pairs(self.cols.x) do
157     s = row[col.at]
158     if x == nil and x == .. then
159         inc = col.like(s,prior)
160         print(inc)
161         like = like + math.log(inc) end end
162 return like end
163
164 function DATA:klass(row) --- return 'row's' class symbol.
165 return (row.cells or row.cells or row[self.self.klass.at]) end
166
167 function DATA:stats(nbec,col,sbo) --- get 'sbo' of 'cols' (round to 'nbdec')
168 cols,sbo = sbo or self.cols.y,sbo or "mid"
169 local t,v
170 t={}
171 for _col in pairs(cols) do
172     v=getmetatable(col["sbo"](col)
173     vtype(v)="number" and rnd(v/nbec) or v
174     [col.txt]=inc end; return t end
175
176 -- ## NB
177 --- ## Update
178 function NB:add(row) --- update the 'datas' about 'row's' klass
179 local kname=new() self.n,self.nb = self.nb; return self.all:clone() end
180 if self.all
181 then self.all:add(row)
182     local k = self.all:klass(row)
183     if #self.all.rows>10 then self.repo[sef.classify(row),k]
184         self.datas[k] = self.datas[k] or new()
185         self.datas[k].add(row)
186     else self.all=DATA(row) end end
187
188 function NB:klass(row) --- which klass likes 'row' the most?
189 local most,klike,e=-math.huge
190 for k,data in pairs(self.datas) do
191     like = data:like(row)
192     if klike > most then most=klike,k=end end
193 return klike end

```

```

208 -
209 -
210 -
211 -
212 - | What | Notes |
213 - +-----+-----+
214 - | adds(data,src) | add list 'src' or filename 'src' to 'data'. |
215 - | cdf(x) | Gaussian cumulative distribution |
216 - | coerce(s) | Parse 'the' config settings from 'help'. |
217 - | copy(i,isShallow,u) | Copy 'i' recursively if if not 'isShallow'. |
218 - | cov($filename, fun) | call 'fun' cells in each CSV line |
219 - | fix(t) | simulate |
220 - | map(t1,fun) | apply 'fun' across 't1' (skip nil results) |
221 - | o(t1,seen,show,u) | push 't1' to string (skip loops, sort slots) |
222 - | push(t,x) | print nested lists |
223 - | end(n,n_places) | Push 'x' to end of 't', return 'x' |
224 - | run(settings,funs) | round 'n' to 'n_places'. |
225 - | settings(s) | run one 'funs', controlled by 'settings'. |
226 - | | create a 'the' variable |
227 - | -| -|
228 -
229 -- ## Math
230 function rnd(n,n_places) --- round 'n' to 'n_places'.
231 local mul := 10^(n_places or 2)
232 return math.floor(n * mul + .5) / mul and
233
234 function pdf(x,mu,sd) --- Gaussian probability distribution
235 return math.exp(-.5*(x-mu)^2/(sd^2*(2*math.pi*.5))) end
236
237 function cdf(x,c,cdf) --- Gaussian cumulative distribution
238 function _cdf(x, p,t) --- Abramowitz and Stegun cdf approximation
239 --- Handbook Mathematical Functions, 1988
240 t = 1 / (1+.0231481x^2)
241 return 1 - p*(0.231838x - 0.356563782*t^2 + 1.781477937*t^3
242 - 1.821255978*t^4 + 1.330274429*t^5)*c
243 and return (x==0 and .5) or (x>0 and _cdf(x)) or 1-_cdf(-x) end
244
245 -- ## Lists
246 function push(x,t) --- Push 'x' to end of 't', return 't'
247 t[t!:=x] := x end
248
249 function map(t1,fun) --- apply 'fun' across 't1' (skip nil results)
250 local t2:=[]; for s1 in pairs(t1) do t2[t2!:=] = fun(s1); end; return t2 end
251
252 function copy(t, isshallow, u) --- copy 't' (recursive if if not 'isshallow')
253 if type(t) == "table" then return t
254 else if for k,v in pairs(t) do u(k) == isshallow and v or copy(v,isshallow) end
255 return setmetatable(u,getmetatable(t)) end
256
257 -- ## Strings & Things
258 function $s($str,..., fun) --- Parse 'the' config settings from 'help'.
259 function fun(s)
260 if s=="w" then return t
261 if s=="false" then return false end
262 return math.tointeger(s) or tonumber(s) or fun(s:match"%$(%$)%") end
263
264 function cov($filename, fun, src,s,t) --- call 'fun' cells in each CSV line
265 src = io.open(src)$filename
266 while true do
267 s = io.read()
268 then t = {}; for sl in smatch("(?![\r\n])") do t[t!:=] = coerce(sl) end
269 fun(t)
270 src:close(src) end end end
271
272 function adds(data,src) --- add list 'src' or filename 'src' to 'data'
273 if type(src)=="string"
274 then csv(src,"table",function(row) data:add(row) end)
275 else map(src={},function(row) data:add(row) end)
276 return data end
277
278 -- ## Thing to String
279 function str(str,...) --- emulate printf
280 return string.format(str,...) end
281
282 function oot(t) --- print nested lists
283 print(o(t)) return t end
284
285 function o(t, seen,show) --- coerce to string (skip loops, sort slots)
286 if type(t) == "table" then return toString(t) end
287 if seen[t] or
288 then return "" end
289 seen[t] = {}
290 function show(k,v)
291 if not toString(k):find("%$") then
292 v = o(v,seen)
293 if show !=>(k and fun("$%$(%$)%")) then v or (v,seen) end
294 else for k,v in pairs(t) do t[t!:=] = show(k,v) end
295 if t[""] then tabln.show(o(t),u)
296 return t[""].table.concat(v,"%$,%$") end
297
298 -- ## Settings
299 function settings(s) --- create a 'the' variable
300 local pat = "(%$pairs)", "%$(%$[%$]?-)?(%$)|(%$)|(%$S)"
301 signmap{pat,function(k,x) t[k]=coerce(x) end}
302 return t end
303
304 function cli(t) --- Updates from command-line. Bool need no values (just flip)
305 for i,j in pairs(t) do
306 v = toString(j)
307 for n,x in pairs(arg) do
308 if x=="-"..(slot:sub(1,i)) or x=="--"..slot then
309 t[i] = "false" and "true" or "true" and "false" or arg[n+1] end end
310 if !he then os.exit(print("Use '-t..help..'")) end
311 return t end
312
313 -- ## Start up
314 function run(settings,funs) --- run one 'funs', controlled by 'settings'
315 local falls,old=0,copy(settings)
316 for k,fun in pairs(funs) do
317 if settings.go == "all" or settings.go == k then
318 for k,v in pairs(old) do settings[srv end
319 math.randomseed(settings.seed or 10019)
320 print("#>>>>>k")
321 if fun(falls) then falls = fails + 1; print("#FALL!!!!",k); end end and
322 for k,v in pairs(LNVD) if not b4{k} then print("#mgmt:",k,type(v)) end

```

```

307 -- (f, f, f, f, f)
308 local go = function()
309   function go.the(i) oo(the) return 1 end
310 end
311
312 function go.cdf(f)
313   for x=2.5,2.5,3 do print(rnd(cdf(x),4), ("="):rep(cdf(x)*50/(1..*)) end
314   end
315   function go.sym sym
316     sym = SYM()
317     for _x in pairs{"a","a","a","a","b","b","b","c"} do sym:append(x)
318     return sym.mode=="a" and sym.most==4 end
319   end
320   function go.num num
321     num = NUM()
322     for x=1,100 do num:append(x) end
323     return 51==rnd(num.mu,0) and 29== rnd(num.sd,0) end
324   end
325   function go.csv()
326     csv(the.file, oo); return 1 end
327   end
328   function go.data(data)
329     data=DATA(the.file)
330     map(data.cols,x,oo) print**
331     map(data.cols,y,oo) end
332   end
333   function go.data(data)
334     data = DATA(the.file)
335     print "mu", o(data:stats(2, data.cols.x,"mid"));
336     print "div", o(data:stats(2, data.cols.x,"div")) end
337   end
338   function go.clone (data,data2)
339     data = DATA(the.file)
340     data2 = data:clone(data1.rows)
341     print "mu", o(data:stats(2, data.cols.x,"mid"));
342     print "mu", o(data2:stats(2, data2.cols.x,"mid")); end
343   end
344   local function _classify(f,nb)
345     local all,correct = 0,0
346     nb=H(f,function(got,want)
347       all=all+1; correct = correct + (got==want and 1 or 0) end
348     print(correct/all) end
349   end
350   function go.diabetes() _classify("Data/diabetes.csv") end
351   function go.soybean() _classify("Data/soybean.csv") end
352 end
353
354 -- ## Start
355 the = settings(heap)
356 if poall(debug.getlocal,4,1)
357   then return (the.the,NUM=NUM,SYM=SYM,DATA=DATA,ROW=ROW,NB=NB)
358   else the=cl(
359     run(the,go) end
360   end

```