


```

152 #
153 #
154 #
155 #
156 #
157 #
158 #
159
160 def any(a:list) -> Any:
161     "Return a random item."
162     return a[anywhere(a)]
163
164 def anywhere(a:list) -> int:
165     "Return a random index of list 'a'."
166     return random.randint(0, len(a)-1)
167
168 big = sys.maxsize
169
170 def atom(x):
171     "Return a number or trimmed string."
172     x=x.strip()
173     if x=="True": return True
174     elif x=="False": return False
175     else:
176         try: return int(x)
177         except: return float(x)
178     except: return x.strip()
179
180
181 def demo(do,all):
182     "Maybe run a demo, if we want it, resetting random seed first."
183     todo = dir(all)
184     if do and do != "all":
185         todo = [x for x in dir(all) if x.startswith(do)]
186     for one in todo:
187         fun = all.__dict__.get(one,"")
188         if type(fun)==type(demo):
189             random.seed(the.seed)
190             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
191             try:
192                 fun()
193                 print("PASS:", doc)
194             except Exception as e:
195                 all.fails += 0
196                 if the.cautious: traceback.print_exc(); exit(1)
197             else:
198                 print("FAIL:", doc, e)
199             exit(all.fails)
200
201 def file(f):
202     "Iterator. Returns one row at a time, as cells."
203     with open(f) as fp:
204         for line in fp:
205             line = re.sub(r'(\n|\r|\v)|#.*', '', line)
206             if line:
207                 yield [atom(cell.strip()) for cell in line.split(",")]
208
209 def first(a:list) -> Any:
210     "Return first item."
211     return a[0]
212
213 def merge(b4:list) -> list:
214     "While we can find similar adjacent things, merge them."
215     j,n,now = -1,len(b4),[]
216     while j < n-1:
217         j += 1
218         a = b4[j]
219         if j < n-2:
220             if merged := a.merge(b4[j+1]):
221                 a = merged
222                 j += 1 # we will continue, after missing one
223             now += [a]
224         # if 'now' is same size as 'b4', look for any other merges.
225     return b4 if len(now)==len(b4) else merge(now)
226
227
228 class o(object):
229     "Class that can pretty print its slots, with fast inits."
230     def __init__(i, **d): i.__dict__.update(**d)
231     def __repr__(i):
232         pre = i.__class__.__name__ if isinstance(i,o) else ""
233         return pre+str(
234             {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})
235
236 def options(doc:str) -> o:
237     "Convert 'doc' to options dictionary using command line args.
238     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
239     those on the command line means 'flip the default value'; (2) args need only
240     mention the first few of a key (e.g. -s is enough to select for -seed).""
241     d={}
242     for line in doc.splitlines():
243         if line and line.startswith(" -"):
244             key, _, x = line.strip()[1:].split("#") # get 1st,last word on each line
245             for j,flag in enumerate(sys.argv):
246                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
247                     x = "True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
248             d[key] = atom(x)
249     if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
250     return o(**d)
251
252 def r() -> float:
253     "Return random number 0..1"
254     return random.random()
255
256 def rn(x:float, n=3) -> float:
257     "Round a number to three decimals."
258     return round(x,n)
259
260 def rN(a:list, n=3) -> list:
261     "Round a list of numbers to three decimals."
262     return [rn(x,n=n) for x in a]
263
264 def second(a:list) -> Any:
265     "Return second item."
266     return a[1]

```

```

265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278
279 class Span(o):
280     """Given two 'Sample's and some 'x' range 'lo..hi'.
281     a 'Span' holds often that range appears in each 'Sample'."""
282     def __init__(i,col, lo, hi, ys=None,):
283         i.col, i.lo, i.hi, i.ys = col, lo, hi, ys or Sym()
284
285     def add(i, x:float, y:Any, inc=1) -> None:
286         "y' is a label identifying one 'Sample' or another."
287         i.lo = min(x, i.lo)
288         i.hi = max(x, i.hi)
289         i.ys.add(y,inc)
290
291     def merge(i, j): # -> Span|None
292         "If the merged span is simpler, return that merge."
293         a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
294         if (i.ys.n==0 or j.ys.n==0 or
295             c.div()*0.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
296             return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)
297
298     def selects(i,row:list) -> bool:
299         "True if the range accepts the row."
300         x = row[i.col.at]; return x=="?" or i.lo<=x and x< i.hi
301
302     def show(i, positive=True) -> None:
303         "Show the range."
304         txt = i.col.txt
305         if positive:
306             if i.lo == i.hi: return f"[txt] == {i.lo}"
307             elif i.lo == -big: return f"[txt] < {i.hi}"
308             elif i.hi == big: return f"[txt] >= {i.lo}"
309             else:
310                 return f"[i.lo] <= [txt] < {i.hi}"
311         else:
312             if i.lo == i.hi: return f"[txt] != {i.lo}"
313             elif i.lo == -big: return f"[txt] >= {i.hi}"
314             elif i.hi == big: return f"[txt] < {i.lo}"
315             else:
316                 return f"[txt] < {i.lo} or [txt] >= {i.hi}"
317
318     def support(i) -> float:
319         "Returns 0..1."
320         return i.ys.n / i.col.n
321
322 @staticmethod
323 def sort(spans : list) -> list:
324     "Good spans have large support and low diversity."
325     divs, supports = Num(), Num()
326     sn = lambda s: supports.norm( s.support())
327     dn = lambda s: divs.norm( s.ys.div())
328     f = lambda s: ((1 - sn(s))*2 + dn(s)**2)**.5/2***.5
329     for s in spans:
330         divs.add( s.ys.div())
331         supports.add(s.support())
332     return sorted(spans, key=f)
333
334 #
335 #
336 #
337
338 class Col(o):
339     "Summarize columns."
340     def __init__(i,at=0,txt=""):
341         i.n,i.at,i.txt,i.w=0,at,txt,(-1 if "-" in txt else 1)
342
343     def dist(i,x:Any, y:Any) -> float:
344         return 1 if x=="?" and y=="?" else i.dist1(x,y)
345
346 #
347 #
348 #
349
350 class Sym(Col):
351     "Summarize symbolic columns."
352     def __init__(i,**kw):
353         super().__init__(**kw)
354         i.has, i.mode, i.mode = {}, None, 0
355
356     def add(i, x:str, inc:int=1) -> str:
357         "Update symbol counts in 'has', updating 'mode' as we go."
358         if x != " ":
359             i.n += inc
360             tmp = i.has[x] = inc + i.has.get(x,0)
361             if tmp > i.mode: i.mode, i.mode = tmp, x
362         return x
363
364     def dist(i,x:str, y:str) ->float:
365         "Distance between two symbols."
366         return 0 if x==y else 1
367
368     def div(i):
369         "Return diversity of this distribution (using entropy)."
370         p = lambda x: x / (1E-31 + i.n)
371         return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )
372
373     def merge(i,j):
374         "Merge two 'Sym's."
375         k = Sym(at=i.at, txt=i.txt)
376         for x,n in i.has.items(): k.add(x,n)
377         for x,n in j.has.items(): k.add(x,n)
378         return k
379
380     def mid(i):
381         "Return central tendency of this distribution (using mode)."
382         return i.mode
383
384     def spans(i,j, out):
385         """For each symbol in 'i' and 'j', count the
386         number of times we see it on either side."""
387         xys = [(x,"this",n) for x,n in i.has.items()] + [
388             (x,"that",n) for x,n in j.has.items()]
389         one, last = None,None
390         all = []
391         for x,y,n in sorted(xys, key=first):
392             if x != last:
393                 last = x
394                 one = Span(i, x,x)
395                 all += [one]
396             one.add(x,y,n)
397             if len(all) > 1 : out += all
398         #
399         #
400

```

```

401 class Num(Col):
402     "Summarize numeric columns."
403     def __init__(i,**kw):
404         super().__init__(**kw)
405         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False
406
407     def add(i,x: float ,inc=1):
408         "Reservoir sampler. If '_all' is full, sometimes replace an item at random."
409         if x != " ":
410             i.n += inc
411             i.lo = min(x,i.lo)
412             i.hi = max(x,i.hi)
413             if len(i._all) < i.max : i.ok=False; i._all += [x]
414             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
415         return x
416
417     def all(i):
418         "Return '_all', sorted."
419         if not i.ok: i.ok=True; i._all.sort()
420         return i._all
421
422     def dist1(i,x,y):
423         if x==" ": y=i.norm(y); x=(1 if y<.5 else 0)
424         elif y==" ": x=i.norm(x); y=(1 if x<.5 else 0)
425         else : x,y = i.norm(x), i.norm(y)
426         return abs(x-y)
427
428     def div(i):
429         """Report the diversity of this distribution (using standard deviation).
430         &pm;2.56.3 &sigma;: is 66.90/95% of the mass. 2&sigma;: So one
431         standard deviation is (90-10)% divide by 2.4 times &sigma;. """
432         return (i.per(.9) - i.per(.1)) / 2.56
433
434     def merge(i,j):
435         "Return two 'Num's."
436         k = Num(at=i.at, txt=i.txt)
437         for x in i._all: k.add(x)
438         for x in j._all: k.add(x)
439         return k
440
441     def mid(i):
442         "Return central tendency of this distribution (using median)."
443         return i.per(.5)
444
445     def norm(i,x):
446         "Normalize 'x' to the range 0..1."
447         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
448
449     def per(i,p:float=.5) -> float:
450         "Return the p-th ranked item."
451         a = i.all(); return a[ int(p*len(a)) ]
452
453     def spans(i,j, out):
454         """Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
455         then count the number of times we the bin on other side.
456         Then merge similar adjacent bins. """
457         lo = min(i.lo, j.lo)
458         hi = max(i.hi, j.hi)
459         gap = (hi-lo) / (6/the.xsmall)
460         xys = [(x,"this",1) for x in i._all] + [(x,"that",1) for x in j._all]
461         one = Span(i.lo,lo)
462         all = [one]
463         for x,y,n in sorted(xys, key=first):
464             if one.hi - one.lo > gap:
465                 one = Span(i, one.hi,x)
466                 all += [one]
467             one.add(x,y,n)
468         all = merge(all)
469         all[0].lo = -big
470         all[-1].hi = big
471         if len(all) > 1: out += all
472
473 #
474 #
475 #
476 #
477 #
478
479 class Explain(o):
480     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
481     def __init__(i,here):
482         i.here, i.span, i.yes, i.no = here, None, None, None
483
484     def show(i,pre=""):
485         if not pre:
486             tmp = i.here.mid(i.here.y)
487             print(f"[{pre:40}]: {len(i.here.rows):5} : {tmp}")
488         if i.yes:
489             s=f"[pre]{i.span.show(True)}"
490             tmp = i.yes.here.mid(i.yes.here.y)
491             print(f"[s:40] : {len(i.yes.here.rows):5} : {tmp}")
492             i.yes.show(pre + "|.")
493         if i.no:
494             s=f"[pre]{i.span.show(False)}"
495             tmp = i.no.here.mid(i.no.here.y)
496             print(f"[s:40] : {len(i.no.here.rows):5} : {tmp}")
497             i.no.show(pre + "|.")
498
499 #
500 #
501 #
502 #
503 #
504 class Cluster(o):
505     "Tree with 'left','right' samples, broken at median between far points."
506     def __init__(i,here,x=None,y=None,c=None,mid=None):
507         i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None
508
509     def show(i,pre=""):
510         s= f"[pre:40] : {len(i.here.rows):5}"
511         print(f"[s] : {i.left} else f[s] : {i.here.mid(i.here.y)}")
512         for kid in [i.left,i.right]:
513             if kid: kid.show(pre + "|.")
514
515 #
516 #
517 #
518 #
519
520 class Sample(o):
521     "Load, then manage, a set of examples."
522     def __init__(i,init=[]):
523         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [],None
524         if str ==type(init): [i.add(row) for row in file(init)]
525         if list==type(init): [i.add(row) for row in init]
526
527     def add(i,a):
528         def col(at,txt):
529             what = Num if txt[0].isupper() else Sym
530             now = what(at=at,txt=txt)
531             where = i.y if "+" in txt or "-" in txt or "!" in txt else i.x
532             if txt[-1] != " ":
533                 where += [now]
534             if "!" in txt: i.klass = now
535             return now
536         #-----
537         if i.cols: i.rows += [(col.add(a[col.at]) for col in i.cols)]
538         else:
539             i.cols = [col(at,txt) for at,txt in enumerate(a)]
540
541     def clone(i,init=[]):
542         out = Sample()
543         out.add([col.txt for col in i.cols])
544         [out.add(x) for x in init]
545         return out
546
547     def cluster(i,top=None):
548         """Split the data using random projections. Find the span that most
549         separates the data. Divide data on that span."""
550         here = Cluster(i)
551         top = top or i
552         if len(i.rows) >= 2*(len(top.rows)**the.enough):
553             left,right,x,y,c,mid = i.half(top)
554             if len(left.rows) < len(i.rows):
555                 here = Cluster(i,x,y,c,mid)
556             here.left = left.cluster(top)
557             here.right = right.cluster(top)
558         return here
559
560     def dist(i,x,y):
561         d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
562         return (d/len(i.x)) ** (1/the.p)
563
564     def div(i,cols=None):
565         return [col.div() for col in (cols or i.all)]
566
567     def far(i, x, rows=None):
568         tmp= sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
569         return tmp[ int(len(tmp)*the.far) ]
570
571     def half(i, top=None):
572         "Using two faraway points 'x,y' break data at median distance."
573         some= i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
574         top= top or i
575         w = any(some)
576         _,x= top.far(w, some)
577         c,y= top.far(x, some)
578         tmp= [r for _,r in sorted([(top.proj(r,x,y,c),r)
579                                     for r in i.rows],key=first)]
580         mid= len(tmp)//2
581         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
582
583     def mid(i,cols=None):
584         return [col.mid() for col in (cols or i.all)]
585
586     def proj(i,row,x,y,c):
587         "Find the distance of a 'row' on a line between 'x' and 'y'."
588         a = i.dist(row,x)
589         b = i.dist(row,y)
590         return (a**2 + c**2 - b**2) / (2*c)
591
592     def xplain(i,top=None):
593         """Split the data using random projections. Find the span that most
594         separates the data. Divide data on that span."""
595         here = Explain(i)
596         top = top or i
597         tiny = len(top.rows)**the.enough
598         if len(i.rows) >= 2*tiny:
599             left, right,_ = i.half(top)
600             spans = []
601             [lcol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
602             if len(spans) > 0:
603                 here.span = Span.sort(spans)[0]
604                 yes, no = i.clone(), i.clone()
605                 [(yes if here.span.selects(row) else no).add(row) for row in i.rows]
606                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
607                 if tiny <= len(no.rows) < len(i.rows): here.no = no.xplain(top=top)
608             return here

```

```

608 #
609 #
610 #
611 #
612 #
613 #
614 #
615
616 class Demos:
617     "Possible start-up actions."
618     fails=0
619     def opt():
620         "show the config"
621         [print(f"{k}>10}={v}") for k,v in the.__dict__.items()]
622
623     def seed():
624         "seed"
625         assert .494 <= r() <= .495
626
627     def num():
628         "check 'Num'."
629         n = Num()
630         for _ in range(100): n.add(r())
631         assert .30 <= n.div() <= .31, "in range"
632
633     def sym():
634         "check 'Sym'."
635         s = Sym()
636         for x in "aaaabbc": s.add(x)
637         assert 1.37 <= s.div() <= 1.38, "entropy"
638         assert 'a' == s.mid(), "mode"
639
640     def rows():
641         "count rows in a file."
642         assert 399 == len([row for row in file(the.data)])
643
644     def sample():
645         "sampling"
646         s = Sample(the.data)
647         assert 398 == len(s.rows), "length of rows"
648         assert 249 == s.x[-1].has[1], "symbol counts"
649
650     def dist():
651         "distance between rows"
652         s = Sample(the.data)
653         assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842
654
655     def far():
656         "distant items"
657         s = Sample(the.data)
658         for _ in range(32):
659             a,_ = s.far(any(s.rows))
660             assert a>.5, "large?"
661
662     def clone():
663         "cloning"
664         s = Sample(the.data)
665         s1 = s.clone(s.rows)
666         d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
667         for k,v in d1.items():
668             assert d2[k] == v, "clone test"
669
670     def half():
671         "divide data in two"
672         s = Sample(the.data); s1,s2,*_ = s.half()
673         print(s1.mid(s1.y))
674         print(s2.mid(s2.y))
675
676     def cluster():
677         "divide data in two"
678         s = Sample(the.data)
679         s.cluster().show(); print("")
680
681     def xplain():
682         "divide data in two"
683         s = Sample(the.data)
684         s.xplain().show(); print("")
685
686 #-----
687 the=options(__doc__)
688 if __name__ == "__main__": demo(the.todo,Demos)
689
690 """
691 all config local to Sample
692 Example class
693 """

```