


```

148 #
149 #
150 #
151 #
152 #
153 #
154 #
155
156 def any(a:list) -> Any:
157     "Return a random item."
158     return a[anywhere(a)]
159
160 def anywhere(a:list) -> int:
161     "Return a random index of list 'a'."
162     return random.randint(0, len(a)-1)
163
164 big = sys.maxsize
165
166 def atom(x):
167     "Return a number or trimmed string."
168     x=x.strip()
169     if x=="True": return True
170     elif x=="False": return False
171     else:
172         try: return int(x)
173         except: return float(x)
174     except: return x.strip()
175
176 def demo(do,all):
177     "Maybe run a demo, if we want it, resetting random seed first."
178     todo = dir(all)
179     if do and do != "all":
180         todo = [x for x in dir(all) if x.startswith(do)]
181     for one in todo:
182         fun = all.__dict__.get(one,"")
183         if type(fun)==type(demo):
184             random.seed(the.seed)
185             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
186             try:
187                 fun()
188                 print("PASS:", doc)
189             except Exception as e:
190                 all.fails += 0
191                 if the.cautious: traceback.print_exc(); exit(1)
192                 else: print("FAIL:", doc, e)
193             exit(all.fails)
194
195 def file(f):
196     "Iterator. Returns one row at a time, as cells."
197     with open(f) as fp:
198         for line in fp:
199             line = re.sub(r'([\n\r\v\ ]|#.*)', '', line)
200             if line:
201                 yield [atom(cell.strip()) for cell in line.split(",")]
202
203 def first(a:list) -> Any:
204     "Return first item."
205     return a[0]
206
207 def merge(b4:list) -> list:
208     "While we can find similar adjacent things, merge them."
209     j,n,now = -1,len(b4),[]
210     while j < n-1:
211         j += 1
212         a = b4[j]
213         if j < n-2:
214             if merged := a.merge(b4[j+1]):
215                 a = merged
216                 j += 1 # we will continue, after missing one
217             now += [a]
218         # if 'now' is same size as 'b4', look for any other merges.
219     return b4 if len(now)==len(b4) else merge(now)
220
221 class o(object):
222     "Class that can pretty print its slots, with fast inits."
223     def __init__(i, **d): i.__dict__.update(**d)
224     def __repr__(i):
225         pre = i.__class__.__name__ if isinstance(i,o) else ""
226         return pre+str(
227             {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})
228
229 def options(doc:str) -> o:
230     """Convert 'doc' to options dictionary using command line args.
231     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
232     those on the command line means 'flip the default value'; (2) args need only
233     mention the first few of a key (e.g. -s is enough to select for -seed)."""
234     d={}
235     for line in doc.splitlines():
236         if line and line.startswith(" -"):
237             key, _, x = line.strip()[1:].split("#") # get 1st,last word on each line
238             for j,flag in enumerate(sys.argv):
239                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
240                     x= "True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
241             d[key] = atom(x)
242     if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
243     return o(**d)
244
245 def r() -> float:
246     "Return random number 0..1"
247     return random.random()
248
249 def rn(x:float, n=3) -> float:
250     "Round a number to three decimals."
251     return round(x,n)
252
253 def rN(a:list, n=3) -> list:
254     "Round a list of numbers to three decimals."
255     return [rn(x,n=n) for x in a]
256
257 def second(a:list) -> Any:
258     "Return second item."
259     return a[1]
260
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 class Span(o):
275     """Given two 'Sample's and some 'x' range 'lo..hi'.
276     a 'Span' holds often that range appears in each 'Sample'."""
277     def __init__(i,col, lo, hi, ys=None,):
278         i.col, i.lo, i.hi, i.ys = col, lo, hi, ys or Sym()
279
280 def add(i, x:float, y:Any, inc=1) -> None:
281     "y' is a label identifying one 'Sample' or another."
282     i.lo = min(x, i.lo)
283     i.hi = max(x, i.hi)
284     i.ys.add(y,inc)
285
286 def merge(i, j): # -> Span|None
287     "If the merged span is simpler, return that merge."
288     a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
289     if (i.ys.n==0 or j.ys.n==0 or
290         c.div()*0.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
291         return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)
292
293 def selects(i,row:list) -> bool:
294     "True if the range accepts the row."
295     x = row[i.col.at]; return x=="?" or i.lo<=x and x<i.hi
296
297 def show(i, positive=True) -> None:
298     "Show the range."
299     txt = i.col.txt
300     if positive:
301         if i.lo == i.hi: return f"[txt] == {i.lo}"
302         elif i.lo == -big: return f"[txt] < {i.hi}"
303         elif i.hi == big: return f"[txt] >= {i.lo}"
304         else: return f"[i.lo] <= [txt] < {i.hi}"
305     else:
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: return f"[txt] < {i.lo} or [txt] >= {i.hi}"
310
311 def support(i) -> float:
312     "Returns 0..1."
313     return i.ys.n / i.col.n
314
315 @staticmethod
316 def sort(spans : list) -> list:
317     "Good spans have large support and low diversity."
318     divs, supports = Num(), Num()
319     sn = lambda s: supports.norm( s.support())
320     dn = lambda s: divs.norm( s.ys.div())
321     f = lambda s: ((1 - sn(s))*2 + dn(s)**2)**.5/2***.5
322     for s in spans:
323         divs.add( s.ys.div())
324         supports.add(s.support())
325     return sorted(spans, key=f)
326
327 #
328 #
329 #
330 #
331 #
332 class Col(o):
333     "Summarize columns."
334     def __init__(i,at=0,txt=""):
335         i.n,i.at,i.txt,i.w=0,at,txt,(-1 if "-" in txt else 1)
336
337 def dist(i,x:Any, y:Any) -> float:
338     return 1 if x=="?" and y=="?" else i.dist1(x,y)
339
340 #
341 #
342 #
343 #
344 #
345 class Sym(Col):
346     "Summarize symbolic columns."
347     def __init__(i,**kw):
348         super().__init__(**kw)
349         i.has, i.mode, i.mode = {}, None, 0
350
351 def add(i, x:str, inc:int=1) -> str:
352     "Update symbol counts in 'has', updating 'mode' as we go."
353     if x != " ":
354         i.n += inc
355         tmp = i.has[x] = inc + i.has.get(x,0)
356         if tmp > i.mode: i.mode, i.mode = tmp, x
357     return x
358
359 def dist(i,x:str, y:str) ->float:
360     "Distance between two symbols."
361     return 0 if x==y else 1
362
363 def div(i):
364     "Return diversity of this distribution (using entropy)."
365     p = lambda x: x / (1E-31 + i.n)
366     return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )
367
368 def merge(i,j):
369     "Merge two 'Sym's."
370     k = Sym(at=i.at, txt=i.txt)
371     for x,n in i.has.items(): k.add(x,n)
372     for x,n in j.has.items(): k.add(x,n)
373     return k
374
375 def mid(i):
376     "Return central tendency of this distribution (using mode)."
377     return i.mode
378
379 def spans(i,j, out):
380     """For each symbol in 'i' and 'j', count the
381     number of times we see it on either side."""
382     xys = [(x,"this",n) for x,n in i.has.items()] + [
383         (x,"that",n) for x,n in j.has.items()]
384     one, last = None,None
385     all = []
386     for x,y,n in sorted(xys, key=first):
387         if x != last:
388             last = x
389             one = Span(i, x,x)
390             all += [one]
391         one.add(x,y,n)
392     if len(all) > 1 : out += all

```

```

393 #
394 #
395 #
396 #
397 class Num(Col):
398     "Summarize numeric columns."
399     def __init__(i,**kw):
400         super().__init__(**kw)
401         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False
402
403     def add(i,x: float ,inc=1):
404         "Reservoir sampler. If '_all' is full, sometimes replace an item at random."
405         if x != "":
406             i.n += inc
407             i.lo = min(x,i.lo)
408             i.hi = max(x,i.hi)
409             if len(i._all) < i.max : i.ok=False; i._all += [x]
410             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
411         return x
412
413     def all(i):
414         "Return '_all', sorted."
415         if not i.ok: i.ok=True; i._all.sort()
416         return i._all
417
418     def dist1(i,x,y):
419         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
420         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
421         else : x,y = i.norm(x), i.norm(y)
422         return abs(x-y)
423
424     def div(i):
425         """"Report the diversity of this distribution (using standard deviation).
426         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
427         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
428         return (i.per(.9) - i.per(.1)) / 2.56
429
430     def merge(i,j):
431         "Return two 'Num's."
432         k = Num(at=i.at, txt=i.txt)
433         for x in i._all: k.add(x)
434         for x in j._all: k.add(x)
435         return k
436
437     def mid(i):
438         "Return central tendency of this distribution (using median)."
439         return i.per(.5)
440
441     def norm(i,x):
442         "Normalize 'x' to the range 0..1."
443         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
444
445     def per(i,p:float=.5) -> float:
446         "Return the p-th ranked item."
447         a = i.all(); return a[ int(p*len(a)) ]
448
449     def spans(i,j, out):
450         """"Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
451         then count the number of times we hit the bin on other side.
452         Then merge similar adjacent bins."""
453         lo = min(i.lo, j.lo)
454         hi = max(i.hi, j.hi)
455         gap = (hi-lo) / (6/the.xsmall)
456         xys = [(x,"this",1) for x in i._all] + [
457             (x,"that",1) for x in j._all]
458         one = Span(i.lo,lo)
459         all = [one]
460         for x,y,n in sorted(xys, key=first):
461             if one.hi - one.lo > gap:
462                 one = Span(i, one.hi,x)
463                 all += [one]
464             one.add(x,y,n)
465         all = merge(all)
466         all[0].lo = -big
467         all[-1].hi = big
468         if len(all) > 1: out += all
469 #
470 #
471 #
472 #
473 #
474
475 class Explain(o):
476     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
477     def __init__(i,here):
478         i.here, i.span, i.yes, i.no = here, None, None, None
479
480     def show(i,pre=""):
481         if not pre:
482             tmp = i.here.mid(i.here.y)
483             print(f"{':40} : {len(i.here.rows):5} : {tmp}")
484         if i.yes:
485             s=f"[pre]{i.span.show(True)}"
486             tmp = i.yes.here.mid(i.yes.here.y)
487             print(f"{s:40} : {len(i.yes.here.rows):5} : {tmp}")
488             i.yes.show(pre + "[. ")
489         if i.no:
490             s=f"[pre]{i.i.span.show(False)}"
491             tmp = i.no.here.mid(i.no.here.y)
492             print(f"{s:40} : {len(i.no.here.rows):5} : {tmp}")
493             i.no.show(pre + "[. ")
494
495 #
496 #
497 #
498 #
499 #
500
501 class Cluster(o):
502     "Tree with 'left','right' samples, broken at median between far points."
503     def __init__(i,here,x=None,y=None,c=None,mid=None):
504         i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None
505
506     def show(i,pre=""):
507         s = f"[pre:40] : {len(i.here.rows):5}"
508         print(f"{s} if i.left else f'{s} : {i.here.mid(i.here.y))")
509         for kid in [i.left,i.right]:
510             if kid: kid.show(pre + "[. ")
511

```

```

510 #
511 #
512 #
513 #
514 #
515
516 class Sample(o):
517     "Load, then manage, a set of examples."
518     def __init__(i,init=[]):
519         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [], None
520         if str == type(inits): [i.add(row) for row in file(inits)]
521         if list == type(inits): [i.add(row) for row in inits]
522
523     def add(i,a):
524         def col(at,txt):
525             what = Num if txt[0].isupper() else Sym
526             now = what(at=at, txt=txt)
527             where = i.y if "+" in txt or "-" in txt or "!" in txt else i.x
528             if txt[-1] != " ":
529                 where += [now]
530             if "+" in txt: i.klass = now
531             return now
532
533         #-----
534         if i.cols: i.rows += [[col.add(a[col.at]) for col in i.cols]]
535         else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
536
537     def clone(i,init=[]):
538         out = Sample()
539         out.add([col.txt for col in i.cols])
540         [out.add(x) for x in inits]
541         return out
542
543     def cluster(i,top=None):
544         """"Split the data using random projections. Find the span that most
545         separates the data. Divide data on that span."""
546         here = Cluster(i)
547         top = top or i
548         if len(i.rows) >= 2*(len(top.rows)**the.enough):
549             left,right,x,y,c,mid = i.half(top)
550             if len(left.rows) < len(i.rows):
551                 here = Cluster(i,x,y,c,mid)
552             here.left = left.cluster(top)
553             here.right = right.cluster(top)
554             return here
555
556     def dist(i,x,y):
557         d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
558         return (d/len(i.x)) ** (1/the.p)
559
560     def div(i,cols=None):
561         return [col.div() for col in (cols or i.all)]
562
563     def far(i, x, rows=None):
564         tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
565         return tmp[ int(len(tmp)*the.far) ]
566
567     def half(i, top=None):
568         "Using two faraway points 'x,y' break data at median distance."
569         some= i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
570         top = top or i
571         w = any(some)
572         _,x= top.far(w, some)
573         c,y= top.far(x, some)
574         tmp = [r for _ ,r in sorted([(top.proj(r,x,y,c),r)
575                                     for r in i.rows],key=first))]
576         mid= len(tmp)//2
577         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
578
579     def mid(i,cols=None):
580         return [col.mid() for col in (cols or i.all)]
581
582     def proj(i,row,x,y,c):
583         "Find the distance of a 'row' on a line between 'x' and 'y'."
584         a = i.dist(row,x)
585         b = i.dist(row,y)
586         return (a**2 + c**2 - b**2) / (2*c)
587
588     def xplain(i,top=None):
589         """"Split the data using random projections. Find the span that most
590         separates the data. Divide data on that span."""
591         here = Explain(i)
592         top = top or i
593         tiny = len(top.rows)**the.enough
594         if len(i.rows) >= 2*tiny:
595             left, right, *_ = i.half(top)
596             spans = []
597             [icol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
598             if len(spans) > 0:
599                 here.span = Span.sort(spans)[0]
600                 yes, no = i.clone(), i.clone()
601                 [yes if here.span.selects(row) else no].add(row) for row in i.rows
602                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
603                 if tiny <= len(no.rows ) < len(i.rows): here.no = no.xplain(top=top)
604             return here
605

```

```

606 #
607 #
608 #
609 #
610 #
611 #
612 #
613
614 class Demos:
615     "Possible start-up actions."
616     fails=0
617     def opt():
618         "show the config"
619         [print(f"{k}>10}={v}") for k,v in the.__dict__.items()]
620
621     def seed():
622         "seed"
623         assert .494 <= r() <= .495
624
625     def num():
626         "check 'Num'."
627         n = Num()
628         for _ in range(100): n.add(r())
629         assert .30 <= n.div() <= .31, "in range"
630
631     def sym():
632         "check 'Sym'."
633         s = Sym()
634         for x in "aaaabbc": s.add(x)
635         assert 1.37 <= s.div() <= 1.38, "entropy"
636         assert 'a' == s.mid(), "mode"
637
638     def rows():
639         "count rows in a file."
640         assert 399 == len([row for row in file(the.data)])
641
642     def sample():
643         "sampling"
644         s = Sample(the.data)
645         assert 398 == len(s.rows), "length of rows"
646         assert 249 == s.x[-1].has[1], "symbol counts"
647
648     def dist():
649         "distance between rows"
650         s = Sample(the.data)
651         assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842
652
653     def far():
654         "distant items"
655         s = Sample(the.data)
656         for _ in range(32):
657             a,_ = s.far(any(s.rows))
658             assert a>.5, "large?"
659
660     def clone():
661         "cloning"
662         s = Sample(the.data)
663         s1 = s.clone(s.rows)
664         d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
665         for k,v in d1.items():
666             assert d2[k] == v, "clone test"
667
668     def half():
669         "divide data in two"
670         s = Sample(the.data); s1,s2,*_ = s.half()
671         print(s1.mid(s1.y))
672         print(s2.mid(s2.y))
673
674     def cluster():
675         "divide data in two"
676         s = Sample(the.data)
677         s.cluster().show(); print("")
678
679     def xplain():
680         "divide data in two"
681         s = Sample(the.data)
682         s.xplain().show(); print("")
683
684 #-----
685 the=options(__doc__)
686 if __name__ == "__main__": demo(the.todo,Demos)
687
688 """
689 all config local to Sample
690 Example class
691 """

```