```
#!/usr/bin/env python3.9
# vim: ts=2 sw=2 sts=2 et :
# autopep8: ignore E20,E401,E226,E302,E41
import re, sys, argparse, itertools
from argparse import ArgumentParser as parse
from argparse import RawTextHelpFormatter as textual
Float = Str = Int = Bool = lambda *l: l[0]

def keys(
  BINS    : Float("bins are of size n**BINS") = .5,
  COLS    : Str("columns to use for inference") = "x",
  DATA    : Str("where to read data") = "../data/auto2.csv",
  EPSILON : Float("small = sd**EPSILON") = .3,
  FAR     : Float("where to look for far things") = .9,
  GOAL    : Str("learning goals: best|rest|other") = "best",
  K       : Int("bayes low class frequency hack") = 2,
  M       : Int("bayes low range frequency hack") = 1,
  P       : Int("distance calculation exponent") = 2,
  SAMPLE  : Int("#samples to find far things?") = 20,
  VERBOSE : Bool("set verbose") = False,
  TOP     : Int("focus on this many") = 20,
  XAMPLE  : Str("egs: '-x ls' lists all, '-x all' runs all") = "" ):
  """
    ,-_|\    (c) Tim Menzies, 2021, unlicense.org.
   /    \    The delta between things is
   \_,-._*    simpler than the things.
        v """

  GOAL = {'best' : lambda b, r: b**2/b+r,
          'rest' : lambda b, r: r**2/(b+r),
          'other': lambda b, r: 1/(b+r)
          }[GOAL]

  # ---------------------------------------------------------------
  class Col(o):
    "Storing columns in 'Col', 'Skip', 'Sym', 'Num'."
    def __init__(i, at=0, txt="", inits=[]):
      i.n, i.at, i.txt = 0, at, txt
      i.w = -1 if "-" in txt else 1
      [i.add(x) for x in inits]

    def add(i, x, n=1):
      if x != "?": i.n += 1; x = i.add1(x, n)
      return x
  # ---------------
  class Skip(Col):
    def add1(i, x, n=1): return x
  # ---------------
  class Sym(Col):
    def __init__(i, **kw): i.has = {}; super().__init__(**kw)

    def add1(i, x, n=1): inc(i.has, x, n); return x

    def bins(i, j):
      for k in (i.has | j.has):
        yield i.has.get(k, 0), True, (i.at, (k, k))
        yield j.has.get(k, 0), False, (j.at, (k, k))

    def dist(i, x, y): return 0 if x == y else 1

    def ent(i):
      return sum(-v/i.n * math.log(v/i.n) for v in i.has.values())

    def merge(i, j):
      k = Sym(at=i.at, txt=i.txt)
      [k.add(x, n) for has in (i.has, j.has) for x, n in has.items()]
      return k

    def merged(i, j):
      k = i.merge(j)
      e1, n1, e2, n2, e, n = i.ent(), i.n, j.ent(), j.n, k.ent(), k.n
      if e1 + e2 < 0.01 or e * .95 < n1 / n * e1 + n2 / n * e2:
        return k
  # ---------------
  class Num(Col):
    def __init__(i, **kw):
      i._all, i.ok = [], False
      super().__init__(**kw)

    def add1(i, x, n):
      x, i.ok = float(x), False
      for _ in range(n): i._all += [x]
      return x

    def all(i):
      if not i.ok: i.ok = True; i._all = sorted(i._all)
      return i._all

    def bins(i, j):
      xy = [(z, True) for z in i._all]+[(z, False) for z in j._all]
      eps = EPSILON * (i.n*i.sd() + j.n*j.sd()) / (i.n + j.n)
      for ((lo, hi), sym) in bins(xy, epsilon=eps, enough=len(xy)**BINS):
        for klass, n in sym.has.items():
          yield n, klass, (i.at, (lo, hi))

    def dist(i, x, y):
      if   x == "?": y = i.norm(y); x = 1 if y < 0.5 else 0
      elif y == "?": x = i.norm(x);y = 1 if x < 0.5 else 0
      else         : x, y = i.norm(x), y.norm(y)
      return abs(x-y)

    def norm(i, x):
      if x == "?": return x
      a = i.all()
      return max(0, min(1, (x-first(a))/(last(a)-first(a)+1E-32)))

    def sd(i): return (per(i.all(), .9) - per(i.all(), .1))/2.56
    def span(i): return (first(i.all()), last(i.all()))
    def wide(i, n=0): return last(i.all()) - first(i.all()) >= n

  #---------------------------------------------------------------
  class Row(o):
    "Data is held in 'Row's which, in turn, are stored  in 'Table's"
    def __init__(i, lst, tab=None): i.tab, i.cells = tab, lst

    def dist(i, j):
      d = n = 1E-32
      for col in i.tab.cols[COLS]:
        n += 1
        x, y = i.cells[at], j.cells[at]
        d += 1 if x == "?" and y == "?" else col.dist(x, y) ** P
      return (d/n) ** (1/P)

    def far(i, rows):
      tmp = [(dist(i, j), j) for _ in range(SAMPLE)]
      return per(sorted(tmp, key=first), FAR)


  # ---------------
  class Table(o):
    def __init__(i, inits=[]):
      i.rows = []
      i.cols = o(all=[], names=[], x=[], y=[], klass=None)
      [i.add(x) for x in inits]

    def add(i, a): i.data(a) if i.cols.names else i.header(a)
    def clone(i, inits=[]): return Table([i.cols.names] + inits)

    def data(i, a):
      a = a.cells if type(a) == Row else a
      a = [col.add(a[col.at]) for col in i.cols.all]
      i.rows += [Row(a, tab=i)]

    def header(i, a):
      i.cols.names = a
      for at, x in enumerate(a):
        new = Skip if i.skipp(x) else (Num if i.nump(x) else Sym)
        new = new(at=at, txt=x)
        i.cols.all += [new]
        if not i.skipp(x):
          i.cols["y" if i.yp(x) else "x"] += [new]
          if i.klassp(x):
            i.cols.klass = new

    def klassp(i, x): return "!" in x
    def nump(i, x): return x[0].isupper()
    def skipp(i, x): return "?" in x
    def yp(i, x): return "-" in x or "+" in x or i.klassp(x)

    #---------------------------------------------------------------
    def bins(xy, epsilon=0, enough=30):
      "Use 'bins' to divide numeric data into ranges."
      def merge(b4):
        j, tmp, n = 0, [], len(b4)
        while j < n:
          a = b4[j]
          if j < n - 1:
            b = b4[j + 1]
            if cy := a.y.merged(b.y):
              a = o(x=(a.x[0], b.x[1]), y=cy)
              j += 1
          tmp += [a]
          j += 1
        return merge(tmp) if len(tmp) < len(b4) else b4

      def divide(xy):
        xy = sorted(xy, key=first)
        bin = o(x=Num(), y=Sym())
        bins = [bin]
        for i, (x, y) in enumerate(xy):
          if bin.x.n >= enough:
            if x != b4 and i < len(xy)-enough and bin.x.wide(epsilon):
              bin = o(x=Num(), y=Sym())
              bins += [bin]
          bin.x.add(x)
          bin.y.add(y)
          b4 = x
        return bins

      return merge(divide([o(bin.x.span(), y=bin.y) for bin in bins]))
```

```
194:
195:   # ---------------
196:   def contrasts(here, there, t):
197:     "Report the ranges that are most different in two classes."
198:     def like(d, kl):
199:       out = prior = (hs[kl] + K) / (n + K*2)
200:       for at, span in d.items():
201:         f = has.get((kl, (at, span)), 0)
202:         out *= (f + M*prior) / (hs[kl] + M)
203:       return out
204:
205:     def val(d): return GOAL(like(d, True), like(d, False)), d
206:     def top(a): return sorted(a, reversed=True, key=first)[:TOP]
207:
208:     has = {(kl, (at, (lo, hi))): f
209:            for col1, col2 in zip(here.cols.x, there.cols.x)
210:            for f, kl, (at, (lo, hi)) in col1.bins(col2)}
211:     n = len(here.rows, there.rows)
212:     hs = {True: len(here.rows), False: len(there.rows)}
213:     solos = [val(dict(at=x)) for at, x in set([z for _, z in has])]
214:     ranges = {}
215:     for _, d in top(solos):
216:       for k in d:
217:         ranges[k] = ranges.get(k, set()).add(d[k])
218:     for rule in top([val(d) for d in dict_product(ranges)]):
219:       print(rule)
220:
221:   #---------------------------------------------------------------
222:   class Eg:
223:     "Unit tests."
224:     def ls():
225:       "list  all examples."
226:       print("\nexamples:")
227:       for k, f in vars(Eg).items():
228:         if k[0] != "_":
229:           print(f"  {k:<13} {f.__doc__}")
230:
231:     def data(file="../data/vote.csv", goal="democrat"):
232:       "simple load of data into  a table"
233:       t = Table(csv(file))
234:       assert(435 == len(t.rows))
235:       assert(195 == t.cols.all[1].has['y'])
236:
237:     def clone(file="../data/diabetes.csv", k="positive"):
238:       "discretize test"
239:       t = Table(csv(file))
240:       kl = t.cols.klass.at
241:       u, v = t.clone(), t.clone()
242:       [(u if k == row.cells[kl] else v).add(row) for row in t.rows]
243:       good, bad = u.cols.x[1], v.cols.x[1]
244:       #print(good.all()[::25], len(good.all()), good.sd())
245:       print(bad.all()[::25], len(bad.all()), bad.sd())
246:
247:     def bins(file="../data/diabetes.csv", k="positive"):
248:       "discretize test"
249:       t = Table(csv(file))
250:       kl = t.cols.klass.at
251:       u, v = t.clone(), t.clone()
252:       print("========================")
253:       for row in t.rows:
254:         (u if k == row.cells[kl] else v).add(row)
255:       good, bad = u.cols.x[1], v.cols.x[1]
256:       for x in good.bins(bad):
257:         print(good.at, x)
258:
259:   # ---------------------------------------------------------------
260:   # main program for keys
261:   if XAMPLE == "all":
262:     for k, f in vars(Eg).items():
263:       if k[0] != "_": print("\n"+k); f()
264:   else:
265:     if XAMPLE and XAMPLE in vars(Eg): vars(Eg)[XAMPLE]()
266:
267:   ##########################################################
268:   # things that don't use the config vars
269:   # dictionaries
270:   def has(d, k): return d.get(k, 0)
271:   def inc(d, k, n=1): tmp = d[k] = n + d.get(k, 0); return tmp
272:   def public(d): return {k: v for k, v in d.items() if k[0] != "_"}
273:
274:   def dict_product(d):
275:     keys = d.keys()
276:     for p in itertools.product(*d.values()):
277:       yield dict(zip(keys, p))
278:
279:   # ------------------
280:   # lists
281:   def first(a): return a[0]
282:   def last(a): return a[-1]
283:   def per(a, p=.5): return a[int(p*len(a))]
284:
285:   # ------------------
286:   class o(object):
287:     "objects"
288:     def __init__(i, **k): i.__dict__.update(**k)
289:     def __getitem__(i, k): return i.__dict__[k]
290:     def __repr__(i): return i.__class__.__name__+str(public(i.__dict__))
291:     def __setitem__(i, k, v): i.__dict__[k] = v
292:
293:   # ------------------
294:   def csv(f=None, sep=","):
295:     "read csv files"
296:     def prep(s): return re.sub(r'([\n\t\r ]|#.*)', '', s)
297:     if f:
298:       with open(f) as fp:
299:         for s in fp:
300:           if s := prep(s): yield s.split(sep)
301:     else:
302:       for s in sys.stdin:
303:         if s := prep(s): yield s.split(sep)
304:
305:   # ------------------
306:   def cli(f):
307:     "Drive command line flags  from function annocations."
308:     p = parse(prog="./"+f.__name__, description=f.__doc__,
309:               formatter_class=textual)
310:     for (k, h),b4 in zip(
311:                       list(f.__annotations__.items()),f.__defaults__):
312:       if b4 == False:
313:         p.add_argument("-"+(k[0].lower()), dest=k, help=h,
314:                        default=False, action="store_true")
315:       else:
316:         p.add_argument("-"+(k[0].lower()), dest=k, default=b4,
317:                        help=h+" ["+str(b4)+"]", type=type(b4),
318:                        metavar=k)
319:     f(**p.parse_args().__dict__)
320:
321:   #---------------------------------------------------------------
322:   if __name__ == "__main__": cli(keys)
```