```python
#!/usr/bin/env python3
# vim: ts=2 sw=2 sts=2 et :
# python3 sublime.py
# (c) 2022, Tim Menzies, unlicense.org",
"""
./sublime.py [OPTIONS]
(c)2022 Tim Menzies unlicense.org

OPTIONS:
  -Max     max numbers to keep           = 512
  -Some    find `far` in this many egs   = 512
  -data    data file                     = ../data/auto93.csv
  -help    show help                     = False
  -far     ihow far to look within `Some` = .9
  -p       distance function coefficient = 2
  -seed    random number seed            = 10019
  -todo    start up task                 = nothing
  -xsmall  Cohen's small effect          = .35
"""
import re,sys,random

# This is free and unencumbered software released into the public domain.
#
# Anyone is free to copy, modify, publish, use, compile, sell, or
# distribute this software, either in source code form or as a compiled
# binary, for any purpose, commercial or non-commercial, and by any
# means.
#
# In jurisdictions that recognize copyright laws, the author or authors
# of this software dedicate any and all copyright interest in the
# software to the public domain. We make this dedication for the benefit
# of the public at large and to the detriment of our heirs and
# successors. We intend this dedication to be an overt act of
# relinquishment in perpetuity of all present and future rights to this
# software under copyright law.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
# IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
# OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
# ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
# OTHER DEALINGS IN THE SOFTWARE.
#
# For more information, please refer to <http://unlicense.org/>


#  ___              __
# /\_ \      __    /\ \
# \//\ \    /\_\   \ \ \____
#   \ \ \   \/\ \   \ \ '__`\
#    \_\ \_  \ \ \   \ \ \L\ \
#    /\____\  \ \_\   \ \_,__/
#    \/____/   \/_/    \/___/
#------------------------------------------------------------------------------


# randoms stuff
r        = random.random
anywhere = lambda a: random.randint(0, len(a)-1)

# useful constants
big      = sys.maxsize

# list membership
first    = lambda a: a[0]
second   = lambda a: a[1]

def atom(x):
  "Return a number or trimmed string."
  x=x.strip()
  if   x=="True" : return True
  elif x=="False": return False
  else:
    try: return int(x)
    except:
      try: return float(x)
      except: return x.strip()

def options(doc):
  d={}
  for line in doc.splitlines():
    if line and line.startswith(" -"):
       key, *_, x = line[3:].split(" ")
       for j,flag in enumerate(sys.argv):
         if flag and flag[0]=="-" and key.startswith(flag[1:]):
            x= "True" if x=="False" else("False" if x=="True" else sys.argv[j+1])
       d[key] = atom(x)
  if d["help"]: exit(print(doc))
  return o(**d)

def demo(want,one,all):
  "Maybe run a demo, if we want it, resetting random seed first."
  if (not want or (want and one.startswith(want))):
    random.seed(the.seed)
    all.__dict__[one]()

def file(f):
  "Iterator. Returns one row at a time, as cells."
  with open(f) as fp:
    for line in fp:
      line = re.sub(r'([\n\t\r"\' ]|#.*)', '', line)
      if line:
        yield [atom(cell.strip()) for cell in line.split(",")]

class o(object):
  "Class that can pretty print its slots, with fast init."
  def __init__(i, **d): i.__dict__.update(**d)
  def __repr__(i): return i.__class__.__name__+str(
      {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})

the = options(__doc__)
```

```python
#           ___
#          /\_ \
#     ___  \//\ \      __      ____    ____     __    ____
#    /'___\  \ \ \   /'__`\   /',__\  /',__\  /'__`\ /',__\
#   /\ \__/   \_\ \_/\ \L\.\_/\__, `\/\__, `\/\  __//\__, `\
#   \ \____\  /\____\ \__/.\_\/\____/\/\____/\ \____\/\____/
#    \/____/  \/____/\/__/\/_/\/___/  \/___/  \/____/\/___/
#--------------------------------------------------------------------------------
class Range(o):
  def __init__(i,col=None,lo=None,hi=None):
    i.col, i.xlo, i.xhi, i.yhas = col, lo, hi, Sym()

  def __add__(i,x,y):
    if x != "?":
      i.lo = min(x,i.lo)
      i.hi = max(x,i.hi)
      i.yhas + y
    return x

  def merge(i,j):
    lo  = math.min(i.lo, j.lo)
    hi  = math.max(i.hi, j.hi)
    z   = 1E-31
    B,R = i.B+z, i.R+z
    k   = Range(i.col, lo, hi, i.b+j.b, i.B, i.r+j.r, j.R)
    if k.b/B < .01 or k.r/R < .01             : return k
    if k.val() > i.val() and k.val() > j.val(): return k

  def __lt__(i,j): return i.val() < j.val()

  def __repr__(i):
    if i.lo == i.hi: return f"{i.col.txt} == {i.lo}"
    if i.lo == -big: return f"{i.col.txt} == {i.hi}"
    if i.hi ==  big: return f"{i.col.txt} >= {i.lo}"
    return f"{i.lo} <= {i.col.txt} < {i.hi}"

  def val(i):
    z=1E-31; B,R = i.B+z, i.R+z; return (i.b/B)**2/( i.b/B + i.r/R)

  def selects(i,row):
    x = row[col.at]; return x=="?" or i.lo<=x and x<i.hi

class Col(o):
  def __init__(i,at=0,txt=""): i.n,i.at,i.txt,i.w = 0,at,txt,(-1 if "<" in txt else 1)
  def __add__(i,x,inc=1):
    if x !="?": i.n += inc; i.add(x,inc)
    return x
  def dist(i,x,y): return 1 if x=="?" and y=="?" else i.dist1(x,y)

class Num(Col):
  def __init__(i,**kw):
    super().__init__(**kw)
    i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False

  def add(i,x,_):
    i.lo = min(x,i.lo)
    i.hi = max(x,i.hi)
    if len(i._all) < i.max    : i.ok=False; i._all += [x]
    elif r()       < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x

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

  def per(i,p=.5):
    a = i.all(); return a[ int(p*len(a)) ]

  def mid(i): return i.per(.5)
  def div(i): return (i.per(.9) - i.per(.1)) / 2.56

  def norm(i,x):
    return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)

  def dist1(i,x,y):
    if   x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
    elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
    else       : x,y = i.norm(x), i.norm(y)
    return abs(x-y)

  def ranges(i,j, all):
    # def merge(b4):
    #   j,n = -1,len(b4)
    #   while j < n:
    #     j += 1
    #     a = b4[j]
    #     if j< n-1:
    #       b=b4[j+1]
    lo  = min(i.lo, j.lo)
    hi  = max(i.hi, j.hi)
    gap = (hi-lo) / (6/the.xsmall)
    at  = lambda z: lo + int((z-lo)/gap)*gap
    all = {}
    for x in map(at, i._all): s=all[x]=(all[x] if x in all else Sym()); s.add(1)
    for x in map(at, j._all): s=all[x]=(all[x] if x in all else Sym()); s.add(0)
    all = merge(sorted(all.items(),key=first))

class Sym(Col):
  def __init__(i,**kw):
    super().__init__(**kw)
    i.has, i.mode, i.most = {}, None, 0

  def add(i,x,inc):
    tmp = i.has[x] = inc + i.has.get(x,0)
    if tmp > i.most: i.most, i.mode = tmp, x

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

  def mid(i): return i.mode
  def div(i):
    p=lambda x: x/i.n
    return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )

  def ranges(i,j, all):
    for x,b in i.has.items(): all += [Range(i,x,x, b,i.n, j.has.get(x,0), j.n)]
    for x,b in j.has.items(): all += [Range(j,x,x, b,j.n, i.has.get(x,0), i.n)]








#--------------------------------------------------------------------------------
class Sample(Col):
  def __init__(i,inits=[]):
    i.rows, i.cols, i.x, i.y = [], [], [], []
    if str ==type(inits): [i + row for row in file(inits)]
    if list==type(inits): [i + row for row in inits]

  def __add__(i,a):
    def col(at,txt):
      what  = Num if txt[0].isupper() else Sym
      now   = what(at=at, txt=txt)
      where = i.y if "+" in txt or "-" in txt or "!" in txt else i.x
      if txt[-1] != ":": where += [now]
      return now
    #-----------
    if i.cols: i.rows += [[col + a[col.at] for col in i.cols]]
    else:      i.cols  = [col(at,txt) for at,txt in enumerate(a)]

  def mid(i,cols=None): return [col.mid() for col in (cols or i.all)]
  def div(i,cols=None): return [col.div() for col in (cols or i.all)]

  def clone(i,inits=[]):
    out = Sample()
    out + [col.txt for col in i.cols]
    [out + x for x in inits]
    return out

  def dist(i,x,y):
    d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
    return (d/len(i.x)) ** (1/the.p)

  def far(i, row1, rows=None):
    tmp= sorted([(i.dist(row1,row2),row2) for row2 in (rows or i.rows)],key=first)
    return tmp[ int(len(tmp)*the.far) ]

  def proj(i,row,x,y,c):
    a = i.dist(row,x)
    b = i.dist(row,y)
    return (a**2 + c**2 - b**2) / (2*c) , row

  def half(i, top=None):
    top  = top or i
    some = random.choices(i.rows, k=the.Some)
    w    = some[0]
    _,x  = top.far(w, some)
    c,y  = top.far(x, some)
    left, right = i.clone(), i.clone()
    for n,(_,r) in enumerate(
                      sorted([top.proj(r,x,y,c) for r in i.rows],key=first)):
      (left if n <= len(i.rows)//2 else right).__add__(r)
    return left,right
```

```python
#   __                                       
#  /\ \                                      
#  \_\ \     __    ___ ___     ___     ____  
#  /'_` \  /'__`\/' __` __`\  / __`\  /',__\ 
# /\ \L\ \/\  __//\ \/\ \/\ \/\ \L\ \/\__, `\
# \ \___,_\ \____\ \_\ \_\ \_\ \____/\/\____/
#  \/__,_ /\/____/\/_/\/_/\/_/\/___/  \/___/ 
#-----------------------------------------------------------------------------
class Egs:
  def num():
    n=Num()
    for i in range(10000): n + i
    print(sorted(n._all),n)

  def sym():
    s=Sym()
    for i in range(10000): s + int(r()*20)
    print(s)

  def rows():
    for row in file(the.data): print(row)

  def sample(): s=Sample(the.data); print(len(s.rows))

  def done(): s=Sample(the.data); s.dist(s.rows[1], s.rows[2])

  def dist():
    s=Sample(the.data)
    for row in s.rows: print(s.dist(s.rows[0], row))

  def far():
    s=Sample(the.data)
    for row in s.rows: print(row,s.far(row))

  def clone():
    s=Sample(the.data); s1 = s.clone(s.rows)
    print(s.x[0])
    print(s1.x[0])

  def half():
    s=Sample(the.data); s1,s2 = s.half()
    print(s1.mid(s1.y))
    print(s2.mid(s2.y))

if __name__ == "__main__":
  for one in dir(Egs): demo(the.todo,one,Egs)
```