```python
#!/usr/bin/env python3 -B
"""
xai.py: explainable multi-objective optimzation
(c) 2025 Tim Menzies, MIT license

Input is CSV. Header (row 1) defines column roles as follows:
  [A-Z]* : Numeric (e.g. "Age").     [a-z]* : Symbolic (e.g. "job").
  *+     : Mximize (e.g. "Pay+").    *-     : Minimize (e.g. "Cost-").
  *X     : Ignored (e.g. "idX").     ?      : Missing value (not in header)

For help on command line options:
  ./xai.py -h

To download example data:
  mkdir -p $HOME/gits
  git clone http://github.com/timm/moot $HOME/gits/moot

To download code, install it, then test it, download this file then:
  chmod +x xai.py
  ./xai.py --xai ~/gits/moot/optimize/misc/auto93.csv"""
import ast,sys,random,re
from math import sqrt,exp,floor
from types import SimpleNamespace as obj
from pathlib import Path

# ATOM = str | int | float
# ROW  = list[ATOM]
# ROWS = list[ROW]
# NUM, SYM, DATA = obj,obj,obj
# COL  = NUM | SYM
# THING = COL | DATA
BIG=1e32
the=obj(bins=7, budget=30, seed=1, leaf=2, check=5, data="data.csv")

### Constructors ----------------------------------------------------------
def Sym(): return obj(it=Sym, n=0, has={})
def Num(): return obj(it=Num, n=0, mu=0, m2=0)

def Col(at=0, txt=" "):
  col = (Num if txt[0].isupper() else Sym)()
  col.at, col.txt, col.best = at, txt, 0 if txt[-1]=="-" else 1
  return col

def Cols(names): # (list[str]) -> Cols
  cols = [Col(n,s) for n,s in enumerate(names)]
  return obj(it=Cols, names=names, all=cols,
             x=[col for col in cols if col.txt[-1] not in "+-X"],
             y=[col for col in cols if col.txt[-1] in "+-"])

def Data(rows=None):
  return adds(rows, obj(it=Data, rows=[], n=0, cols=None, _centroid=None))

def clone(data, rows=None): return adds(rows, Data([data.cols.names]))

### Update ----------------------------------------------------------------
def adds(src, i=None): # (src:Iterable, ?i) -> i
  i = i or Num(); [add(i,v) for v in src or []]; return i

def add(i, v, inc=1):
  if v!="?":
    if Data is i.it and not i.cols: i.cols = Cols(v) # init, not adding
    else:
      i.n += inc # adding
      if   Sym is i.it: i.has[v] = inc + i.has.get(v,0)
      elif Num is i.it:
        if inc < 0 and i.n < 2:
          i.mu = i.m2= i.n=0
        else:
          d = v-i.mu; i.mu += inc*d/i.n; i.m2 += inc*d*(v-i.mu)
      else:
        i._centroid = None # old centroid now out of date
        [add(col, v[col.at], inc) for col in i.cols.all] # recursive add
        (i.rows.append if inc>0 else i.rows.remove)(v)   # row storage
  return v # convention: always return the thing being added

### Queries ---------------------------------------------------------------
def norm(num,n):
  z = (n - num.mu) / sd(num)
  z = max(-3, min(3, z))
  return 1 / (1 + exp(-1.7 * z))

def sd(num): return 1/BIG + (0 if num.n<2 else sqrt(max(0,num.m2)/(num.n-1)))

def mid(col): return col.mu if Num is col.it else max(col.has,key=col.has.get)

def mids(data):
  data._centroid = data._centroid or [mid(col) for col in data.cols.all]
  return data._centroid

def disty(data,row):
  ys = data.cols.y
  return sqrt(sum(abs(norm(y,row[y.at]) - y.best)**2 for y in ys) / len(ys))

def distx(data,row1,row2):
  xs = data.cols.x
  return sqrt(sum(_aha(x, row1[x.at], row2[x.at])**2 for x in xs) / len(xs))

def _aha(col,u,v):
  if u==v=="?": return 1
  if Sym is col.it : return u != v
  u,v = norm(col,u), norm(col,v)
  u = u if u != "?" else (0 if v>0.5 else 1)
  v = v if v != "?" else (0 if u>0.5 else 1)
  return abs(u - v)


```

```python
## Cutting ----------------------------------------------------------------
def Cut(at,txt,lo,hi):
  return obj(it=Cut, at=at, txt=txt, xlo=lo, xhi=hi, y=Num())

def cutShow(cut, accept=True):
  s,lo,hi = cut.txt, cut.xlo, cut.xhi
  if lo == hi:
    return f"{s} {'==' if accept else '!='} {lo}"
  if hi == BIG:
    return f"{s} {'>=' if accept else '<'} {lo}"
  if lo == -BIG:
    return f"{s} {'<' if accept else '>='} {hi}"
  return f"{lo} <= {s} < {hi}" if accept else f"{s} < {lo} or {s} >= {hi}"

def cutSelects(cut, row):
  if (x:=row[cut.at]) == "?" : return True
  if cut.xlo == cut.xhi      : return x == cut.xhi
  return cut.xlo <= x < cut.xhi

def cutScore(cut):
  if cut.y.n<the.leaf: return BIG
  return cut.y.mu + sd(cut.y) / (sqrt(cut.y.n) + 1/BIG)

def cutRows(data, rows):
  all_bins = (b for col in data.cols.x for b in cutsRows(col, rows, data))
  return min(all_bins, key=lambda b: cutScore(b), default=None)

def cutsRows(col, rows, data):
  d, xys = {}, [(r[col.at], disty(data, r)) for r in rows if r[col.at]!="?"]
  for x, y in sorted(xys):
    k = x if Sym is col.it else floor(the.bins * norm(col, x))
    if k not in d:
      d[k] = Cut(col.at,col.txt, x, x)
    add(d[k].y, y)
    d[k].xhi = x
  return cutsComplete(col, sorted(d.values(), key=lambda b: b.xlo))

def cutsComplete(col, cuts):
  if Num is col.it:
    for n, b in enumerate(cuts):
      b.xlo = cuts[n-1].xhi if n > 0 else -BIG
      b.xhi = cuts[n+1].xlo if n < len(cuts)-1 else BIG
  return cuts

## Trees ------------------------------------------------------------------
# Trees recursively cut data.
def Tree(n, mu, mids, cut):
  return obj(it=Tree, n=n, mu=mu, mids=mids, cut=cut, kids={})

def treeGrow(data, rows=None, cut=None):
  rows = rows or data.rows
  tree = Tree(len(rows),
              disty(data,mids(clone(data,rows))),
              mids(clone(data,rows))[len(data.cols.x)+1:],
              cut)
  if len(rows) > the.leaf*2:
    if cut1 := cutRows(data,rows):
      ok,no = [],[]
      for row in rows: (ok if cutSelects(cut1,row) else no).append(row)
      if ok and no:
        tree.kids[True]  = treeGrow(data, ok, cut1)
        tree.kids[False] = treeGrow(data, no, cut1)
  return tree

def treeShow(tree, lvl=0,accept=True,width=60,dec=1):
  if lvl==0: print(" ")
  here = f"{cutShow(tree.cut,accept)} " if lvl>0 else "."
  report = f"{'| ' * (lvl-1)}{here}"
  print(f"{report:{width}}: {o(tree.mu):6}: {tree.n:>4} : ",
        ','.join([f"{n:.{dec}f}"for n in tree.mids]))
  for k, kid in tree.kids.items():
    treeShow(kid, lvl + 1,k,width,dec)

def treeLeaf(tree, row):
  if tree.kids:
    rule = tree.kids[True].cut
    return treeLeaf(tree.kids[cutSelects(rule, row)], row)
  return tree


## Lib --------------------------------------------------------------------
def gauss(mid,div):
  return mid + 2 * div * (sum(random.random() for _ in range(3)) - 1.5)

def o(v=None, DEC=2,**D):
  if D: return o(D,DEC=DEC)
  isa = isinstance
  if isa(v, (int, float)): return f"{round(v, DEC):_}"
  if isa(v, list):  return f"[{', '.join(o(k,DEC) for k in v)}]"
  if isa(v, tuple): return f"({', '.join(o(k,DEC) for k in v)})"
  if callable(v):   return v.__name__
  if hasattr(v, "__dict__"): v = vars(v)
  if isa(v, dict): return "{"+ " ".join(f":{k} {o(v[k],DEC)}" for k in v) +"}"
  return str(v)

def coerce(s):
  try: return int(s)
  except Exception as _:
    try: return float(s)
    except Exception as _:
      s=s.strip()
      return {"true":True, "false":False}.get(s,s)

def csv(fileName):
  with open(fileName,encoding="utf-8") as f:
    for l in f:
      if (l:=l.split("%")[0].strip()):
        yield [coerce(x) for x in l.split(",")]

def shuffle(lst): random.shuffle(lst); return lst


```

```
#-----------------------------------------------------------------------------
def go_h(_=None):
  ": show help"
  print(__doc__,"\n\nOptions:\n")
  for k,f in globals().items():
    if k.startswith("go_") and f.__doc__:
      left, right = f.__doc__.split(":")
      left = k[2:].replace("_","-") + " " + left.strip()
      d = f.__defaults__
      default = f"(default: {d[0]})" if d else ""
      print(f" {left:15}  {right.strip()} {default}")

def go_b(n=the.bins):
  "INT : set number of BINS used on discretization"
  the.bins = n

def go_B(n=the.budget):
  "INT : set BUDGET for rows labelled each round"
  the.budget = n

def go_c(n=the.check):
  "INT : set numer of evals for final check"
  the.check = n

def go_l(n=the.leaf):
  "INT : set minimum exampels per leaf"
  the.leaf = n

def go_s(n=the.seed):
  "INT : set random number seed"
  the.seed = n; random.seed(n)

def go__all(file=the.data):
  "FILE : run all actions that use a FILE"
  for k,fun in globals().items():
    if k.startswith("go__") and k != "go__all":
      go_s(1)
      print("\n#",k,"------------"); fun(file)

def go__num(_=None):
  ": test Nums"
  num = adds(gauss(10, 2) for _ in range(1000))
  print(o(mu=num.mu, sd=sd(num)))
  assert 9.9 <= num.mu <=10.1 and 1.9 <= sd(num) <= 2.1

def go__sym(_=None):
  ": test Syms"
  sym = adds('Previously, we have defined an iterative data mining',Sym())
  print(sym.has)
  assert sym.has["a"]==5

def go__csv(file=the.data):
  "FILE : test csv loading"
  total=0
  for n,row in enumerate(csv(file)):
    if n > 0: total += len(row)
    if n > 0: assert isinstance(row[1], (float,int))
    if n % 40==0: print(row)
  assert 3184 == total

def go__data(file=the.data):
  "FILE : test ading columns from file"
  data =  Data(csv(file))
  total = sum(len(row) for row in data.rows)
  print(*data.cols.names)
  assert Num is data.cols.all[0].it
  assert 3184 == total
  for col in data.cols.x: print(o(col))

def go__clone(file=the.data):
  "FILE : test echoing structure of a table to a new table"
  data1 =  Data(csv(file))
  data2 = clone(data1,data1.rows)
  assert data1.cols.x[1].mu == data2.cols.x[1].mu

def go__distx(file=the.data):
  "FILE : can we sort rows by their distance to one row?"
  data=Data(csv(file))
  print(*data.cols.names,"distx",sep=",")
  r1 = data.rows[0]
  data.rows.sort(key=lambda r2: distx(data,r1,r2))
  for n,r2 in enumerate(data.rows[1:]):
    assert 0 <= distx(data, r1,r2) <= 1
    if n%40==0: print(*r2,o(distx(data,r1,r2)),sep=",")

def go__disty(file=the.data):
  "FILE : can we sort rows by their distance to heaven?"
  data=Data(csv(file))
  print(*data.cols.names,"disty",sep=",")
  data.rows.sort(key=lambda r: disty(data,r))
  for n,r1 in enumerate(data.rows):
    if n>0:
      r2=data.rows[n-1]
      assert disty(data, r1) >= disty(data,r2)
    if n%40==0: print(*r1,o(disty(data, r1)),sep=",")

def go__bins(file=the.data):
  "FILE : show the rankings of a range"
  data = Data(csv(file))
  all_bins = (b for col in data.cols.x for b in cutsRows(col, data.rows, data))
  for b in sorted(all_bins, key=lambda b: cutScore(b)):
    print(f"{cutShow(b):20}", o(mu=b.y.mu, sd=sd(b.y), n=b.y.n,
                              scored= cutScore(b)),sep="\t")

def go__xai(file=the.data, repeats=1):
  data = Data(csv(file))
  b4   = sorted([disty(data,row) for row in data.rows])
  lo   = b4[0]
  mid  = b4[len(b4)//2]
  Y    = lambda row: disty(data,row)
  win  = lambda row: int(100*(1- (Y(row) - lo)/ (mid - lo + 1/BIG)))
  nums = Num()
  wins = Num()
  n    = len(data.rows)//2
  for _ in range(repeats):
    rows = shuffle(data.rows)
    test = rows[n:]
    train= clone(data,rows[:n][:the.budget-the.check])
    tree = treeGrow(train)
    if repeats == 1: treeShow(tree,width=35)
    X = lambda row: treeLeaf(tree,row).mu
    guess = min(sorted(test,key=X)[:the.check],key=Y)
    if repeats==1:
      print(o(lo=lo, mid=mid, guess=Y(guess), win=win(guess)))
    else:
      add(nums, Y(guess))
      add(wins, win(guess))
  if repeats>1:
    print(o(wins=wins.mu, n=nums.n, lo=lo, mid=mid, guess=o(nums.mu)),
         re.sub(r".*/","",file))
```

```

def go__xais(file=the.data): go__xai(file,repeats=20)

if __name__ == "__main__":
  go_s(1)
  for n, s in enumerate(sys.argv):
    if fn := vars().get(f"go{s.replace('-', '_')}"):
      fn(coerce(sys.argv[n+1])) if n < len(sys.argv) - 1 else fn()
```