## xai4.py Page 1/4

```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)

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

For help on command line options:
  ./xai.py -h """
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=50, seed=1, 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]))

### Functions ------------------------------------------------------------
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        i._centroid = None # old centroid now out of date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

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)

## Cutting ----------------------------------------------------------------
def score(num): return num.mu + sd(num) / (sqrt(num.n) + 1/BIG)

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

def cut(data, rows):
  all_bins = (b for col in data.cols.x for b in cuts(col, rows, data))
  return min(all_bins, key=lambda b: score(b.y), default=None)

def cuts(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] = obj(at=col.at, txt=col.txt, xlo=x, xhi=x, y=Num())
    add(d[k].y, y)
    d[k].xhi = x
  return _complete(col, sorted(d.values(), key=lambda b: b.xlo))

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

## xai4.py Page 2/4

```python

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

def o(v=None, DEC=3,**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_s(n=the.seed):
  "INT : set random SEED "
  the.seed = n; random.seed(the.seed)

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__all(file=the.file):
  "FILE : run all actions that use a FILE"
  for k,fun in globals().items():
    if k.startswith("go__") and k != "go__all":
      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.file):
  "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.file):
  "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.file):
  "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.file):
  "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.file):
  "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.file):
  "FILE : show the rankings of a range"
  data = Data(csv(file))
  all_bins = (b for col in data.cols.x for b in cuts(col, data.rows, data))
  for b in sorted(all_bins, key=lambda b: score(b.y)):
    print(b.txt,b.xlo,b.xhi, o(mu=b.y.mu, sd=sd(b.y), n=b.y.n,
                               scored= score(b.y)),sep="\t")

def go__xai(file=the.file):
  "FILE : can we succinctly list main effects in a table?"
  print("\n"+re.sub(r"^.*/","",file))
  xai(Data(csv(file)))

def xai(data,rows=None,loud=True):
  if loud:
    print("x :",len(data.cols.x))
    print("y :",len(data.cols.y))
    print("r :",len(data.rows))
    print("b :",the.bins)
  def goals(data,row): return [row[goal.at] for goal in data.cols.y]
  if loud: print(*goals(data,data.cols.names),sep=",")
  def show(n): return "-\u221e" if n==-BIG else "\u221e" if n==BIG else o(n)
  def go(rows, lvl=0, prefix=""):
    ys = Num(); rows.sort(key=lambda row: add(ys, disty(data, row)))
    if loud:
      print(f"{o(goals(data,mids(clone(data,rows)))),: {o(mu=ys.mu, n=ys.n, sd=sd(ys)):25s} {prefix}")
    if rule := cut(data, rows):
      rules.append(rule)
      now = [row for row in rows if select(rule, row)]
      if 2 < len(now) < len(rows):
        txt = rule.xlo if rule.xlo==rule.xhi \
                       else f"[{show(rule.xlo)} .. {show(rule.xhi)})"
        return go(now, lvl + 1, f"{rule.txt} is {txt}")
    return rules,rows
  rules=[]
  return go(rows or data.rows, 0)

def go__lurch(file=the.file):
  "FILE : can we succinctly list main effects in a table using random selection?"
  print("\n"+re.sub(r"^.*/","",file))
  data = Data(csv(file))
  ninety,few,br=Num(),Num(),Num()
  Y= lambda row: disty(data,row)
```



```
  def learn(train,test):
     labelled=clone(data,train)
    _,best= xai(labelled,loud=False)
    bmid = mids(clone(data,best))
    return sorted(test, key=lambda row: distx(labelled,row,bmid))
  def poles(train,test):
    train.sort(key=lambda row: disty(data,row))
    n=int(sqrt(len(train)))
    bmid,rmid = mids(clone(data, train[:n])), mids(clone(data,train[n:]))
    seen=clone(data,train)
    return sorted(test, key=lambda r: distx(seen, r,bmid)- distx(seen,r,rmid))
  def check(rows): return Y(min(rows[:5], key=Y))
  for _ in range(20):
    rows   = shuffle(data.rows)
    train1 = rows[:int(0.9*len(rows))]
    train2 = rows[:the.budget]
    test   = rows[len(rows)//2:]
    add(ninety, check(learn(train1,test)))
    add(few,    check(learn(train2,test)))
    add(br,     check(poles(train2,test)))
  all = adds(Y(row) for row in data.rows)
  print("b4",o(mu=all.mu,sd=sd(all)),sep="\t")
  print("90%",o(mu=ninety.mu,sd=sd(ninety)),sep="\t")
  print(f"rules{the.budget+5}",o(mu=few.mu,sd=sd(few)),sep="\t")
  print("br" ,o(mu=br.mu,sd=sd(br)),sep="\t")

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()
```