```python
#!/usr/bin/env python3
"""
bins.py: stochastic landscape analysis for multi objective reasoning
(c) 2025 Tim Menseedzies, <timm@ieee.org>. MIT license

Options, with (defaults):

  -b    bins    set bins (5)
  -d    dims    set dimensions (5)
  -f    file    data name (../moot/optimize/misc/auto93.csv)
  -p    p       set mankowski coeffecient (2)
  -s    seed    set random number seed (123456781)
  -S    Some    a few rows to explore (128)
 """
import traceback,random,math,sys,re
sys.dont_write_bytecode = True

pick = random.choice
picks = random.choices
big = 1E32

class o:
  __init__= lambda i, **d: i.__dict__.update(**d)
  __repr__= lambda i: cat(i.__dict__)

#-------------------------------------------------------------------------
def adds(i, src): [add(i,x) for x in src]; return i

def Num(inits=[],at=0, txt=" "):
  return adds(o(it=Num, at=at, txt=txt, n=0, m2=0,mu=0,sd=0, hi= -big, lo= big,
                goal= 0 if txt[-1] == "-" else 1), inits)

def Sym(inits=[], at=0, txt=" "):
  return adds(o(it=Sym, at=at, txt=txt, n=0, has={}, nump=False), inits)

def Cols(names):
  all,x,y = [],[],[]
  for c,s in enumerate(names):
    all += [(Num if s[0].isupper() else Sym)(at=c, txt=s)]
    if s[-1] != "X":
      (y if s[-1] in "+-" else x).append(all[-1])
  return o(it=Cols, all=all, x=x, y=y)

def Data(inits):
  inits=iter(inits)
  return adds( o(it=Data, n=0, _rows=[], cols=Cols(next(inits))), inits)

def clone(data, rows=[]):
  return adds(data(), [data.names] + rows)

def add(i, v):
  def _data(data,row):
    data._rows += [row]
    [add(col, row[col.at]) for col in data.cols.all]

  def _sym(sym,x): sym.has[x] = 1 + sym.has.get(x,0)

  def _num(num,n):
    num.lo  = min(n, num.lo)
    num.hi  = max(n, num.hi)
    d       = n - num.mu
    num.mu += (d / num.n)
    num.m2 += (d * (n - num.mu))
    num.sd  = 0 if num.n < 2 else (num.m2/(num.n - 1))**0.5

  if v != "?":
    i.n = i.n + 1
    (_num if i.it is Num else (_sym if i.it is Sym else _data))(i,v)
  return v

#-------------------------------------------------------------------------
def norm(num,v):
  return v if v=="?" else (x - num.lo) / (num.hi - num.lo + 1/big)

def minkowski(src):
  d, n = 0, 1/big
  for x in src:
    n += 1
    d += x**the.p
  return (d / n)**(1 / the.p)

def ydist(data, row):
  return minkowski(abs(norm(c, row[c.at]) - c.goal) for c in data.cols.y)

def ysort(data,rows=None):
  return sorted(rows or data._rows, key=lambda row: ydist(data,row))

def xdist(data, row1, row2):
  def _aha(col,u,v):
    if u=="?" and v=="?": return 1
    if col.it is Sym: return u!=v
    u = norm(col,u)
    v = norm(col,v)
    u = u if u != "?" else (0 if v > .5 else 1)
    v = v if v != "?" else (0 if u > .5 else 1)
    return abs(u - v)
  return minkowski(_aha(c, row1[c.at], row2[c.at]) for c in data.cols.x)

#-------------------------------------------------------------------------
def project(data,row,a,b):
  X = lambda r1,r2: xdist(data,r1,r2)
  c = xdist(data,a,b)
  return 0 if c==0 else (X(row,a)^2 + c^2 - X(row,b)^2) / (2*c*c)

def bucket(data,row,a,b):
  return min(int( project(data,row,a,b) * the.bins), the.bins - 1)

def extrapolate(data,row,a,b):
  ya, yb = ydist(data,a), ydist(data,b)
  return ya + project(data,row,a,b) * (yb - ya)

def corners(data):
  r0, *some = picks(data._rows, k=the.Some + 1)
  out = [max(some, key=lambda r1: xdist(data,r1, r0))]
  for _ in range(the.dims):
    out += [max(some, key=lambda r2: sum(xdist(data,r1,r2) for r1 in out))]
  return out

def buckets(data, crnrs):
  out = {}
  for row in data._rows:
    k = tuple(bucket(data,row, a, b) for a, b in zip(crnrs, crnrs[1:]))
    out[k] = out.get(k) or clone(data)
    add(out[k], row)
  minPts = 2 if data.n < 100 else max(4, 2*the.Dims)
  return {k:data for k,data in out.items() if data.n >= minPts}

def neighbors(a, bckts):
  return [b for b in bckts if all((abs(m,n) <= 1) for m,n in zip(a,b))]

#-------------------------------------------------------------------------
def atom(x):
  for what in (int, float):
    try: return what(x)
    except Exception: pass
  x = x.strip()
  y = x.lower()
  return (y == "true") if y in ("true", "false") else x

def csv(file):
  with open(file, 'r', newline='', encoding='utf-8') as f:
    for line in f:
      if line:
        yield [atom(s) for s in line.strip().split(',')]

def cli(d):
  for k, v in d.items():
    for c, arg in enumerate(sys.argv):
      if arg == "-" + k[0]:
        d[k] = atom("False" if str(v) == "True" else (
                    "True" if str(v) == "False" else (
                    sys.argv[c + 1] if c < len(sys.argv) - 1 else str(v))))

def run(fn,x=None):
  try: random.seed(the.seed); fn(x)
  except Exception as e:
    return [print(s.strip()) for s in traceback.format_tb(e.__traceback__)]

def cat(v):
  it = type(v)
  inf = float('inf')
  if it is list:  return "{" + ", ".join(map(cat, v)) + "}"
  if it is float: return str(int(v)) if -inf<v<inf and v==int(v) else f"{v:.3g}"
  if it is dict:  return cat([f":{k} {cat(w)}" for k, w in v.items()])
  if it in [type(abs), type(cat)]: return v.__name__
  return str(v)

#-------------------------------------------------------------------------
def eg_h(_):
  ":    show help"
  print("\n"+__doc__.strip())
  for s,fn in globals().items():
    if s.startswith("eg_"):
      print(f" {s[2:].replace("_","-"):6s} {fn.__doc__[1:]}")

def eg__the(_):
  ":    show config"
  print(the)

def eg__data(_):
  data = Data(csv(the.file))
  print(data.n)
  [print("y",col) for col in data.cols.y]

#-------------------------------------------------------------------------
the = o(**{m[1]: atom(m[2])
           for m in re.finditer(r"-\w+\s+(\w+)[^\()*\(\s*([^)]+)\s*\)", __doc__)})

if __name__ == "__main__":
  cli(the.__dict__)
  for i,s in enumerate(sys.argv):
    if fn := globals().get("eg" + s.replace("-", "_")):
      run(fn, None if i == len(sys.argv) - 1 else atom(sys.argv[i+1]))
```