```python
#!/usr/bin/env python3 -B
# vim: ts=2:sw=2:sts=2:et
"""
binr.py : build rules via stochastic incremental XAI
(c) 2025, Tim Menzies, timm@ieee.org, mit-license.org

Options:

  -h          Show help.
  -b  bins=4     Number of bins for discretization (int).
  -B  Budget=30  Max rows to eval (int).
  -e  era=10     Number of rows in an era (int)
  -p  p=2        Distance coefficient
  -r  repeats=20 Number of experimental repeats (int).
  -s  seed=42    Random number seed (int).
  -f  file=../data/auto93.csv File to load (str).
"""
from math import floor,sqrt,cos,log,exp,pi
from typing import Any,Iterable
import fileinput,random,sys,re
rand = random.random

class o(dict):
  "Structs with slots accessiable via x.slot. And pretty print."
  def __repr__(i): return "{" + ' '.join(f":{k} {show(i[k])}" for k in i) + "}"
  def __setattr__(i, k, v): i[k] = v
  def __getattr__(i, k):
    try: return i[k]
    except KeyError: raise AttributeError(k)

the = o(bins=4, Budget=30, era=10, p=2, repeats=20, seed=42,
        file="../data/auto93.csv")

Qty  = float | int
Atom = Qty | str | bool
Row  = list[Atom]
Rows = list[Row]
# Num,Sym,Cols = o,o,o       # defined below
# Col  = Num | Sym           # defined below
# Data = tuple[Rows, Cols]   # defined below

# ---------------------------------------------------------------------------
def Sym() -> o:
  "Summarize symbol."
  return o(it=Sym, n=0, has={}, bins={})

def Num() -> o:
  "Summarize numbers."
  return o(it=Num, n=0, mu=0, sd=0, m2=0, bins={})

def Col(at=0, of=" ") -> o:
  "Column in rows of data."
  it = (Num if of[0].isupper() else Sym)()
  it.at = at
  it.of = of
  it.best = str(of)[-1]!="-"
  return it

def Cols(names:list[str]) -> o:
  "Factory. Turns column names into columns."
  cols = [Col(at=i, of=s) for i,s in enumerate(names)]
  return o(it=Cols, names=names,
           all = cols,
           x   = [col for col in cols if str(col.of)[-1] not in "+-X"],
           y   = [col for col in cols if str(col.of)[-1] in "+-"])

def Data(rows = None) -> o:
  "Summarize rows into columns."
  return adds(rows, o(it=Data, n=0, rows=[], cols=None))

# ---------------------------------------------------------------------------
def add(i: o, # o = Col | Data,
        item: Any,
        inc = 1) -> Any: # returns item
  "Add or subtract items from columns or data."
  if item=="?": return item
  i.n += inc
  if   i.it is Sym: i.has[item] = inc + i.has.get(item,0)
  elif i.it is Num:
    item = float(item)
    if inc < 0 and i.n < 2:
      i.n = i.mu = i.sd = i.m2 = 0
    else:
      d     = item - i.mu
      i.mu += inc * d / i.n
      i.m2 += inc * d * (item - i.mu)
      i.sd  = 0 if i.n < 2 else sqrt(max(0,i.m2)/(i.n - 1))
  elif i.it is Data:
    if i.cols:
      row = [add(c, item[c.at], inc) for c in i.cols.all]
      i.rows.append(row) if inc > 0 else i.rows.remove(row)
    else: i.cols = Cols(item)
  return item

def sub(i,item):
  "Subtract items."
  return add(i,item,-1)

def adds(items:Iterable = None, it=None ) -> o: # returns it
  "Load many items into ‘it‘ (defeault is ‘Num()‘)."
  it = it or Num()
  if str(items)[-4:]==".csv":
    with open(items, encoding="utf-8") as f:
      for line in f:
        if line: add(it, [s.strip() for s in line.split(",")])
  else: [add(it, item) for item in (items or [])]
  return it


```

```python
# ---------------------------------------------------------------------------
def norm(num:Num, v:Qty) -> float:
  "Returns 0..1."
  return 1/(1+exp(-1.702 * (v- num.mu)/(num.sd + 1e-32))) if v != "?" else v

def bin(col:Col, v:Atom) -> int | Atom:
  "Returns 0..bins-1."
  return floor( the.bins * norm(col,v) ) if v!="?" and col.it is Num else v

def dist(src:Iterable) -> float:
  "Mankoski distance."
  d,n = 0,0
  for d1 in src:
    n += 1
    d += d1 ** the.p
  return (d/n) ** (1/the.p)

def disty(data:Data, row:Row) -> float:
  "Distance of ‘row‘ to ‘best‘ values in each goal column."
  return dist(abs(norm(col, row[col.at]) - col.best) for col in data.cols.y)

def distx(data:Data, row1:Row, row2:Row) -> float:
  "Distance between ‘x‘ attributes of two rows."
  return dist(_aha(col, row1[col.at], row2[col.at]) for col in data.cols.x)

def _aha(col:Col, a:Atom, b:Atom) -> float:
  "If any unknowns, assume max distance."
  if a==b=="?": return 1
  if col.it is Sym : return a != b
  a,b = norm(col,a), norm(col,b)
  a = a if a != "?" else (0 if b>0.5 else 1)
  b = b if b != "?" else (0 if a>0.5 else 1)
  return abs(a - b)

# ---------------------------------------------------------------------------
def scoreGet(use, row:Row) -> Row:
  "Sum the score of the bins used by ‘row‘."
  n = 0
  for num in use:
    if (v := row[num.at]) != "?":
      print(v, num, bin(num,v))
      if bin(num, v) == num.of:
        n += want(num)
        print(22)
  return n

def scorePut(data:Data, row:Row, score:Qty):
  "Increment the bins used by ‘row‘."
  for x in data.cols.x:
    if (b := bin(x, row[x.at])) != "?":
      one = x.bins[b] = x.bins.get(b) or Num()
      one.at, one.of = x.at, b
      add(one, score)

def want(num): return num.mu + num.sd/sqrt(num.n)

def top(data):
  return sorted((num for x in data.cols.x for num in x.bins.values()),key=want)

def score(data:Data, eps=0.05):
  "Guess next few scores using scores seen to date."
  best_score, best_row = 1e32, None
  random.shuffle(data.rows)
  seen, rows, model = set(), data.rows, Data([data.cols.names])
  for j, row in enumerate(rows):
    if len(seen) >= the.Budget: break
    add(model, row)
    scorePut(model, row, disty(model, row))
    seen.add(id(row))
    if (j+1) % the.era == 0 and j < len(rows) - 100:
      use = top(model)[:5]
      candidate = min(rows[j+1:j+20], key=lambda r: scoreGet(use, r))
      seen.add(id(candidate))
      if (score := disty(model, candidate)) < best_score - eps:
        best_score, best_row = score, candidate
  return best_score


```

```python
# ---------------------------------------------------------------------------
def show(x):
  "Pretty print."
  if type(x) is type(show) : return x.__name__ + '()'
  if type(x) is float : return str(int(x)) if x == int(x) else f"{x:.2f}"
  return str(x)

# ---------------------------------------------------------------------------
def test_h(_) -> None:
  print(__doc__)

def test__the(_) -> None:
  print(the)

def test_s(n: str) -> None:
  the.seed = float(n); random.seed(the.seed)

def test__sym(_) -> None:
  print(adds("aaaabbc",Sym()))

def test__num(_) -> None:
  def boxMuller(mu,sd): return mu + sd * sqrt(-2*log(rand())) * cos(2*pi*rand())
  print(adds(boxMuller(10,2) for _ in range(10**4)))

def test__data(f = None) -> None:
  data = Data(f or the.file)
  print(data.cols.x[-1])
  print(len(data.rows),data.rows[1])

def test__disty(f = None):
  ys, data = Num(), Data(f or the.file)
  Y=lambda row: floor(100*disty(data,row))
  for r in sorted(data.rows,key=Y)[::20]:
    print(Y(r),r)

def test__distx(f = None):
  xs, data = Num(), Data(f or the.file)
  X=lambda row1: floor(100*distx(data,row1, data.rows[0]))
  for r in sorted(data.rows,key=X)[::20]:
    print(X(r),r)

def test__score(f= None):
  my   = lambda n: floor(100*n)
  data = Data(f or the.file)
  print(len(data.rows))
  ys   = adds(my(disty(data,row)) for row in data.rows)
  print(o(mu=ys.mu,sd=ys.sd))
  print(*sorted(my(score(data)) for _ in range(the.repeats)))

_tests= {k:fun for k,fun in vars().items() if "test__" in k}

def test__all(_):
  for k,fun in _tests.items(): print("\n----- "+k); fun(_)

# ---------------------------------------------------------------------------
if __name__ == "__main__":
  for n, s in enumerate(sys.argv):
    if fn := vars().get(f"test{s.replace('-', '_')}"):
      random.seed(the.seed)
      fn(sys.argv[n+1] if n < len(sys.argv)-1 else None)
```