```
#   <img src="bingo.png" width=200 align=left>
# This code reads csv data from `-f file`, then divides those rows into
# `-B Bins`  along `-d dimes` random projections.
#
#  After randomly scoring
# `-a a` bins, then `-b b` times, it selects two labeled examples,
# guesses their  y-values via extrapolation, then labels the best guess.
#
# Afterwards, `-c c` items from the top bain are labeled for evaluation.
# Those results are compared against a regression tree built from the
# `a+b` examples. This code is successful if the tree finds rows just
# as good as anything else, after just labeling a few rows.
#
# #### In this code:
# - `the` is config, parsed from top docstring (can be updated via CLI);
# - `_` marks private vars/methods;
# - `i` means `self`;
# - `col` means `num` or `sym`, often shortenned to `c`.
# - `row` = `List[int|num|str]` asdasas
# - vars called `d,a,n,s` are often dictionary, array, number, string;
# - structs use `struct.it` to denote type;
# - struct constructors are functions starting with uppercase; e.g. `Sym.,Num`
# - stuct variables are named after their constructor. e.g. `sym,num1`
# - no classes (so polymorphic methods can stay together in the source).
# - `eg__xxx` are CLI demos (run with `--xxx`);
# - The input data is csv,  where row one names the column;  e.g.
# ```
# name    , ShoeSize, Age+
# tim     ,  12     ,   50
# junjie ,   5     ,  100
# ...     ,  ...    ,  ...
# ```
# In row1, upper case names denote numeric columns. Names ending with `+`, `-` a
re
# the `y` goals  to be maximized/minimize. Other columns are the
# `x` independent variables. The input data has all the `y` values known, but th
at
# is just for testing purposes. The core `bingo` algorithm only ever glances at
# a handful of those labels.
"""
bingo.py: stochastic landscape analysis for multi objective reasoning
(c) 2025 Tim Menzies, <timm@ieee.org>. MIT license

Options, with their (defaults):

  -B Bins   number of bins (10)
  -d dims   number of dimensions (4)
  -p p      minkowski coefficient  (2)
  -a a      rows labelled at random during cold start (4)
  -b b      rows labelled while reflecting on labels seen so far (30)
  -c c      rows labels while testing the supposed best bin (5)
  -f file   csv file for data (../../moot/optimize/misc/auto93.csv)
  -k k      Bayes hack for rare classes  (1)
  -m m      Bayes hack for rare frequencies (2)
  -r rseed  random number seed (1234567890)
  -z zero   ignore bins with zero items; 0=auto choose (0)
  -h        show help

"""
from pprint import pformat as say
import urllib.request, random, math, sys, re, os

sys.dont_write_bytecode = True
pick=random.choice
picks=random.choices

### Command-line  ---------------------------------------------------------

# Reset slots from CLI flags, matching on first letter of slot.
# e.g. `-f file1` sets `d["file"]` to `file1`. If current value is a bolean then
# flags reverse old value. e.g. `-v `negates  (e.g.) `d["verbose"]=False`.
def cli(d):
  for k, v in d.items():
    for c, arg in enumerate(sys.argv):
      if arg == "-" + k[0]:
        d[k] = coerce("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))))

# String to thing
def coerce(x):
  for what in (int, float):
    try: return what(x)
    except: pass
  x = x.strip()
  y = x.lower()
  return (y == "true") if y in ("true", "false") else x

def eg_h():
  "print help text"
  print(__doc__,"\nExamples:")
  for s,fun in globals().items():
    if s.startswith("eg__"):
      print(f" {re.sub('eg__','--',s):>6}   {fun.__doc__}")

def eg__all():
  "run all examples"
  for s,fun in globals().items():
    if s.startswith("eg__"):
      if s != "eg__all":
        print(f"\n# {s} {"-"*40}\n# {fun.__doc__}\n")
        random.seed(the.rseed)
        fun()

### Settings  -------------------------------------------------------------
# Structs with named fields + pretty print.
class o:
  __init__= lambda i, **d: i.__dict__.update(**d)
  __repr__= lambda i: \
            (f.__name__ if (f:=i.__dict__.get("it")) else "")+say(i.__dict__)

# Parse the `__doc__` string to generate `the` config variable.
the= o(**{m[1]: coerce(m[2])
          for m in re.finditer(r"-\w+\s+(\w+)[^\[]*\[\s*([^]]+)\s*\]", __doc__)})

def eg__the() -> None:
  "Print the configuration."
  print(the)


### Create ----------------------------------------------------------------
# Update `it` with  multiple things.
def inits(things, it): [add(it,thing) for thing in things]; return it

# Summarize a stream of numbers
def Num(init=[], txt=" ",at=0): # -> Num
  return inits(init,
               o(it=Num,
                 n=0,      # count of items
                 at=at,    # column position
                 txt=txt,  # column name
                 mu=0,     # mean of what what seen
                 _m2=0,    # second moment (used to find sd)
                 lo =-BIG, # lowest seen
                 hi =BIG,  # largest
                 heaven=(0 if txt[-1]=="-" else 1))) # 0,1 = minimize,maximize

# Summarize a stream of symbols
def Sym(init=[], txt=" ",at=0):  # -> Sym
  return inits(init, o(it=Sym,n=0,     # count of items
                       at=at,   # column position
                       txt=txt, # column name
                       has={})) # hold symbol counts

# Turn column names into columns (if upper case, then `Num`. Else `Sym`).
def Cols(names): # -> Cols
  all,x,y = [],[],[]
  for c,s in enumerate(names):
    all += [(Num if s[0].isupper() else Sym)(txt=s,at=c)]
    if s[-1] != "X": # what to ignore
      (y if s[-1] in "+-" else x).append(all[-1])
  return o(it=Cols,all=all, # all the columns
                   x=x,     # just the x columns
                   y=y)     # just the y columns

# Keep some `rows`, summarize them in the `cols`.
def Data(init=[]): # -> Data
  init = iter(src)
  names = next(init) # column names
  return inits(init, o(it=Data, rows = [],           # contains the rows
                                cols = Cols(names))) # summaries of the rows

# Mimic the structure of an existing `Data`. Optionally, add some rows.
def clone(data, rows=[]): # -> Data
  return adds(Data([[col.txt for col in data.cols.all]]), rows)


```

```
### Update -----------------------------------------------------------------
# `sub` is just `add`ing -1.
def sub(i,v,purge=False): # -> v
  return add(iv, n= -1, purge=purge)

# If `v` is unknown, then ignore. Else, update.
def add(i,v, n=1, purge=False): # -> v
  def _sym(sym,s): # update symbol counts
    sym.has[s] = n + sym.has.get(s,0)

  def _data(data,row): # keep the new row, update the cols summaries.
    if n < 0:
      if purge: data.rows.remove(v)
      [sub(col, row[col.at], col) for col in data.cols.all]
    else:
      data.rows += [[add(col, row[col.at]) for col in data.cols.all]]

  def _num(num,n): # update lo,hi, mean and _m2 (used in sd calculation)
    num.lo = min(n, num.lo)
    num.hi = max(n, num.hi)
    if n < 0 and num.n < 2:
      num._m2 = num.mu = num.n = 0
    else:
      d         = n - num.mu
      num.mu  += n * (d / num.n)
      num._m2 += n * (d * (v -   num.mu))

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


### Reports ----------------------------------------------------------------
def mids(data): return [mid(col) for col in data.cols.all]

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

def div(col):
  def _num(num):
    return (max(num.m2,0)/(num.n - 1))**0.5

  def _sym(sym):
    return -sum(v/sym.n * math.log(v/sym.n, 2) for v in sym.has.values() if v>0)

  return (_num if i.it is Num else _sym)(col)

### Bayes ------------------------------------------------------------------
def like(data, row, nall=2, nh=100):
  n = len(data.rows)
  prior = (n + the.k) / (nall + the.k*nh)
  tmp = [pdf(c,row[c.at], prior, nall, nh)
         for c in i.cols.x if row[c.at] != "?"]
  return sum(math.log(n) for n in tmp + [prior] if n>0)

def pdf(col,v, prior=0, nall=2, nh=100):
  def _sym(sym,s):
    return (sym.has.get(s,0) + the.m*prior) / (n + the.m + 1/BIG)

  def _num(num,n):
    sd = num.div() or 1 / BIG
    var = 2 * sd * sd
    z = (x - num.mu) ** 2 / var
    return min(1, max(0, math.exp(-z) / (2 * math.pi * var) ** 0.5))

  return (_num if i.it is Num else _sym)(col,v)
```

```
### Distance ------------------------------------------------------------
def norm(i,v):
  return v if (v=="?" or i.it is not Num) else (v - i.lo)/(i.hi - i.lo + 1/BIG)

def dist(col,v,w):
  def _sym(sym,s1,s2):
    return s1 != s2

  def _num(num,n1,n2):
    n1,n2 = norm(num,n1), norm(num,n2)
    n1 = n1 if n1 != "?" else (0 if n2 > 0.5 else 1)
    n2 = n2 if n2 != "?" else (0 if n1 > 0.5 else 1)
    return abs(n1 - n2)

  return 1 if v=="?" and w=="?" else (_num if i.it is Num else _sym)(col,v,w)

def minkowski(a):
  total, n = 0, 1 / BIG
  for x in a:
    n += 1
    total += x**the.P
  return (total / n)**(1 / the.P)

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

def xdist(data, row1, row2):
  return minkowski(dist(c,row1[c.at], row2[c.at]) for c in data.cols.x)
```

```
### Clustering ----------------------------------------------------------
def project(data, row, a, b): # -> 0,1,2 .. the.bins-1
  D = lambda row1,row2: xdist(data,row1,row2)
  c = D(a,b)
  if c==0: return 0
  return (D(row, a)**2 + c**2 - D(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 poles(data): # -> List[Row]
  r0, *some = picks(i.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 lsh(data, poles): # -> Dict[Tuple, List[Row]]
  buckets = {}
  for row in data.rows:
    k = tuple(bucket(row, a, b) for a, b in zip(poles, poles[1:]))
    buckets[k] = buckets.get(k) or clone(data)
    add(buckets[k], row)
  return buckets

def neighbors(c, hi):
  def go(i, p):
    if i == len(c):
      t = tuple(p)
      if t != c and all(0 <= x < hi for x in t):
        yield t
    else:
      for d in [-1, 0, 1]:
        yield from go(i+1, p + [c[i] + d])
  yield from go(0, [])
```

```
### Tree -------------------------------------------------------------------
ops = {'<=' : lambda x,y: x <= y,
       "==" : lambda x,y: x == y,
       '>'  : lambda x,y: x >  y}

def selects(row, op, at, y): x=row[op]; return  x=="?" or ops[op](x,y)

def cuts(col,rows,Y,Klass):
  def _sym(sym):
    n,d = 0,{}
    for row in rows:
      x = row[i.at]
      if x != "?":
        n = n + 1
        d[x] = d.get(x) or Klass()
        add(d[x], Y(row))
    return o(div = sum(c.n/n * div(c) for c in d.values()),
             hows = [("==",c.at,k) for k,v in d.items()])

  def _num(num):
    out, b4, lhs, rhs = None, None, Klass(), Klass()
    xys = [(r[i.at], add(rhs, Y(r))) for r in rows if r[i.at] != "?"]
    xpect = div(rhs)
    for x, y in sorted(xys, key=lambda xy: x[0]):
      if x != b4:
        if the.leaf <= lhs.n <= len(xys) - the.leaf:
          tmp = (lhs.n * div(lhs) + rhs.n * div(rhs)) / len(xys)
          if tmp < xpect:
            xpect, out = tmp, [("<=", i.at, b4), (">", i.at, b4)]
      add(lhs, sub(rhs,y))
      b4 = x
    if out:
      return o(div=xpect, hows=out)

  return (_sym if col.it is Sym else _num)(col)

def tree(data1, rows=None, Klass=Num, how=None):
  Y          = lambda row: ydist(data1,row)
  rows       = rows or i.rows
  data2.kids = []
  data2.how  = how
  data2      = clone(data1, rows)
  data2.ys   = Num(Y(row) for row in rows)
  if len(rows) >= the.leaf:
    cuts = [tmp for c in t.cols.x if (tmp := cuts(c,rows,Y,Klass=Klass))]
    if cuts:
      for how in sorted(cuts, key=lambda cut: cut.div)[0].hows:
        rows1 = [row for row in rows if selects(row, *how)]
        if the.leaf <= len(rows1) < len(rows):
          data2.kids += [tree(data1, rows1, Klass=Klass, how=how)]
  return data2

def nodes(data1, lvl=0, key=None):
  yield lvl, data1
  for data2 in (sorted(data1.kids, key=key) if key else data1.kids):
    yield from nodes(data2, lvl + 1, key=key)

def leaf(data1,row):
  for data2 in data1.kids or []:
    if selects(row, *data2.decision):
      return leaf(data2, row)
  return data1

def show(data, key=lambda z:z.ys.mu):
  stats = i.ys
  win = lambda x: 100-int(100*(x-stats.lo)/(stats.mu - stats.lo))
  print(f"{'d2h':>4} {'win':>4} {'n':>4} ")
  print(f"{'----':>4} {'----':>4} {'----':>4} ")
  for lvl, node in nodes(data, key=key):
    leafp = len(node.kids)==0
    post = ";" if leafp else ""
    xplain = ""
    if lvl > 0:
      op,at,y = node.decision
      xplain = f"{data.cols.all[at].txt} {op} {y}"
    print(f"{node.ys.mu:4.2f} {win(node.ys.mu):4} {len(node._rows):4}   {(lvl-1) * '| '}{xplain}" + post)


### Utils -------------------------------------------------------------------
def moot(fn):
  if fn.startswith("MOOT/"):
    cdir = os.path.expanduser("~/tmp/moot/")
    os.makedirs(cdir, exist_ok=True)
    lfn = fn[len("MOOT/"):]
    lpath = os.path.join(cdir, lfn)
    if not os.path.exists(lpath):
      rurl = f"https://github.com/timm/moot/{fn}"
      urllib.request.urlretrieve(rurl, lpath)
    return lpath

def csv(s):
  with open(moot(s) or s, 'r', newline='') as f:
    for line in f:
      yield [coerce(s) for s in line.strip().split(',')]

def cat(v):
  it = type(v)
  if it is list:  return "{" + ",".join(map(cat, v)) + "}"
  if it is float: return str(int(x)) if v == int(v) else f"{x:.3g}"
  if it is dict:  return cat([f":{k} {cat(w)}" for k, w in v.items()])
  return say(v)
```

```
### Start-up ------------------------------------------------------------
if __name__ == "__main__":
  cli(the.__dict__)
  for n, s in enumerate(sys.argv):
    if fun := globals().get("eg" + s.replace("-", "_")):
      random.seed(the.rseed)
      fun()
```