```
# &nbsp; <img src="bingo.png" width=250 style="padding-left:20px;" align=right>
# `bingo.py` reads a  CSV file (`-f`) then
# (1) bins the rows into `-B` Bins along `-d` random projections; then
# (2) actively learns by scoring `-a` random bins, then for `-b` iterations,
# extrapolates from 2 examples to label best `y`-guess; then
# (3) `-c` top bin items are labeled for evaluation.
#
# Success here means that  trees learned from (from `a+b` labels) finds stuff
# as good as anything else (after seeing very few labels).
#
# ### Coding conventions:
# - `the` is for CLI config,
# - `row` is a list,
# - prefix `_` means "private",
# - `col` or `c` means `Num` or `Sym`,
# - `i` means "self",
# - `d,a,n,s` = dict, array, num, str,
# - Structs (no classes), since polymorphic code can be shown together,
# - `.it` in structs denotes type
# - Uppercase functions are constructors (e.g. `Sym`), and their matching
#   lowercase names are variables from that constructor (e.g. `sym`),
# - `eg__xxx` are CLI demos (e.g. `--xxx`),
# - In CSV input files, uppercase names on row1 denotes numeric; `+`/`-`
# - show `y`-goals (others `x`).
"""
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)
  –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)
  –d dims   number of dimensions (4)
  –f file   csv file for data (../moot/optimize/misc/auto93.csv)
  –G Got    directory to cache downloaded data files (~/tmp/moot)
  –g get    github repo storing example data files (timm/moot)
  –K Ksee   sample size, when seeking centroids (256)
  –k k      Bayes hack for rare classes  (1)
  –m m      Bayes hack for rare frequencies (2)
  –p p      minkowski coefficient  (2)
  –r rseed  random number seed (1234567891)
  –z zero   ignore bins with zero items; 0=auto choose (0)
  –h        show help
"""
import urllib.request, random, math, sys, re, os

sys.dont_write_bytecode = True
pick = random.choice
picks = random.choices
BIG = 1E32

### 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 Exception: 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 "")+cat(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 `i` with  multiple things.
def inits(things, i): [add(i,thing) for thing in things]; return i

# 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
  cols,x,y = [],[],[]
  for c,s in enumerate(names):
    cols += [(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(cols[-1])
  return o(it=Cols,all=cols, # 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(init)
  names = next(init) # column names
  return inits(init, o(it=Data, n=0,
                                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 inits(Data([[col.txt for col in data.cols.all]]), rows)
```

```
### Read --------------------------------------------------------------------

# Iterate over rows in file `s`.
def csv(s):
  with open(webdata(s) or s, 'r', newline='', encoding='utf-8') as f:
    for line in f:
      yield [coerce(s) for s in line.strip().split(',')]

# Get data from repo `the.get`. Keep a local cache of gotten files at `the.Got`
def webdata(fn):
  if fn.startswith(the.get):
    cdir = os.path.expanduser(the.Got)
    os.makedirs(cdir, exist_ok=True)
    lfn = fn[len(the.get)+1:]
    lpath = os.path.join(cdir, lfn)
    if not os.path.exists(lpath):
      rurl = f"https://github.com/{the.get}/tree/master/{fn}"
      urllib.request.urlretrieve(rurl, lpath)
    return lpath

def eg__csv():
  "Print csv data."
  m = 0
  for n,row in enumerate(csv(the.file)):
    if n>0: assert int is type(row[0])
    m += len(row)
    if n%50==0: print(n,row)
  assert m==3192

def eg__cols():
  "Print csv data."
  cols = (lbs,acc,mpg) = Cols( next(csv(the.file))).y
  assert mpg.heaven==1 and lbs.heaven==0 and acc.at==6
  [print(cat(col)) for col in cols]


### Update --------------------------------------------------------------------

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

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

  def _data(data,row): # keep the new row, update the cols summaries.
    if inc < 0:
      if purge: data.rows.remove(v)
      [sub(col, row[col.at], inc) for col in data.cols.all]
    else:
      data.rows += [[add(col, row[col.at],inc) 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 inc < 0 and num.n < 2:
      num._m2 = num.mu = num.n = 0
    else:
      d        = n - num.mu
      num.mu  += inc * (d / num.n)
      num._m2 += inc * (d * (n - num.mu))

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

def eg__num():
  "Demo Numerics."
  g=lambda: random.gauss(10,2)
  num = Num(g() for _ in range(256))
  assert 10 < mid(num) < 10.05 and 2 < div(num) < 2.1

def eg__sym():
  "Demo Symbolics."
  sym = Sym("aaaabbc")
  assert mid(sym) == "a" and 1,37 < div(sym) < 1.38

def eg__data():
  "Read data from disk."
  model = Data(csv(the.file)).cols.x[2]
  assert 3.69 <div( model) < 3.7
  assert model.lo == 70 and model.hi == 82
```

```python
### Reports -------------------------------------------------------------------

def mids(data): return [mid(col) for col in data.cols.all]
def divs(data): return [div(col) for col in data.cols.all]

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

def eg__addSub():
  head, *rows = list(csv(the.file))
  data = Data([head])
  for row in rows:
    add(data,row)
    if data.n == 50: m0,d0 = mids(data),divs(data)
  for row in rows[::-1]:
    sub(data,row)
    if data.n == 50:
      m1,d1 = mids(data), divs(data)
      assert all(math.isclose(a,b,rel_tol=0.01) for a,b in zip(m0, m1))
      assert all(math.isclose(a,b,abs_tol=0.01) for a,b in zip(d0, d1))

### Bayes ----------------------------------------------------------------------

def like(data, row, nall=2, nh=100):
  prior = (data.n + the.k) / (nall + the.k*nh)
  tmp = [pdf(c,row[c.at],prior)
         for c in data.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):
  def _sym(sym,s):
    return (sym.has.get(s,0) + the.m*prior) / (col.n + the.m + 1/BIG)

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

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

def eg__bayes():
  data = Data(csv(the.file))
  L = lambda r: round(like(data,r),2)
  F = lambda a: print(’ ’.join([f"{x:>8}" for x in a]))
  assert all(-20 < L(row) < -9 for row in data.rows)
  rows = [[L(row)] + row for row in sorted(data.rows, key=L)[::30]]
  head = ["Like"] + [col.txt for col in data.cols.all]
  report(rows,head,1)


### 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(_,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 col.it is Num else _sym)(col,v,w)

# Returns the i‘p-‘th root of sum of the x in a (rarraised to ‘p‘).
def minkowski(a):
  total, n = 0, 1 / BIG
  for x in a:
    n += 1
    total += x**the.p
  return (total / n)**(1 / the.p)

# Distance to ideal, measured across y-columns.
def ydist(data, row):
  return minkowski(abs(norm(c,row[c.at]) - c.heaven) for c in data.cols.y)

# Distance between two rows, measured across x-columns.
def xdist(data, row1, row2):
  return minkowski(dist(c,row1[c.at], row2[c.at]) for c in data.cols.x)

# K-means plus plus: k points, usually D^2 distance from each other.
def kpp(data, k=10, rows=None, few=None):
    def D(x, y):
      key = tuple(sorted((id(x), id(y))))
      if key not in mem: mem[key] = xdist(data,x,y)
      return mem[key]

    few = few or the.Ksee
    row,  *rows    = shuffle(rows or data.rows)
    some, rest     = rows[:few], rows[few:]
    centroids, mem = [row], {}
    for _ in range(1, k):
      dists = [min(D(x, y)**2 for y in centroids) for x in some]
      r     = random.random() * sum(dists)
      for j, d in enumerate(dists):
        r -= d
        if r <= 0:
          centroids.append(some.pop(j))
          break
    return centroids, mem, some + rest

def eg__ydist():
  data = Data(csv(the.file))
  L = lambda r: round(like(data,r),2)
  Y = lambda r: round(ydist(data,r),2)
  assert all(0 <= Y(row) <= 1 for row in data.rows)
  rows = [[Y(row),L(row)] + row for row in sorted(data.rows, key=Y)[::30]]
  head = ["Y","Like"] + [col.txt for col in data.cols.all]
  report(rows,head,1)

def eg__kpp():
  "Diversity sample: random vs kpp. Try a few times  with –r $RANDOM ––kpp."
  data = Data(csv(the.file))
  repeats=20
  Y = lambda row: ydist(data,row)
  best = lambda rows: Y(sorted(rows, key=Y)[0])
  b4 = Num(Y(row) for row in data.rows)
  print("b4   ", o(Ksee=len(data.rows), repeats=1, lo=b4.lo, mu=b4.mu, hi=b4.hi))
  for k in [10,20,30,40,80,160]:
    print("")
    anys = Num(best(picks(data.rows,k=k))    for _ in range(repeats))
    print("random ", o(Ksee=k, repeats=anys.n, lo=anys.lo, mu=anys.mu, hi=anys.hi, D=0.35*div(anys)))
    kpps = Num(best(kpp(data,        k=k)[0]) for _ in range(repeats))
    print("kpps  ", o(Ksee=k, repeats=kpps.n, lo=kpps.lo, mu=kpps.mu, hi=kpps.hi, D=0.35*div(kpps)))
```

```
### 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(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 lsh(data, corners): # -> Dict[Tuple, List[Row]]
  buckets = {}
  for row in data.rows:
    k = tuple(bucket(row, a, b) for a, b in zip(corners, corners[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[at]; return  x=="?" or ops[op](x,y)

def cuts(col,rows,Y,Klass):
  def _sym(sym):
    n,d = 0,{}
    for row in rows:
      if (x := row[sym.at]) != "?":
        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 = [("==",sym.at,k) for k,v in d.items()])

  def _num(num):
    out, b4, lhs, rhs = None, None, Klass(), Klass()
    xys = [(r[num.at], add(rhs, Y(r))) for r in rows if r[num.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, [("<=", num.at, b4), (">", num.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(data, Klass=Num, Y=None, how=None):
  Y         = Y or (lambda row: ydist(data,row))
  data.kids = []
  data.how  = how
  data.ys   = Num(Y(row) for row in data.rows)
  if data.n >= the.leaf:
    tmp = [x for c in data.cols.x if (x := cuts(c,data.rows,Y,Klass=Klass))]
    if tmp:
      for how1 in sorted(tmp, key=lambda cut: cut.div)[0].hows:
        rows1 = [row for row in data.rows if selects(row, *how1)]
        if the.leaf <= len(rows1) < data.n:
          data.kids += [tree(clone(data,rows1), Klass, Y, how1)]
  return data

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 = data.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} {node.n:4}   {(lvl-1) * ’| ’}{xplain}" + post)


```

```
### Utils ----------------------------------------------------------------------

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 report(rows, head, decs=2):
  w=[0] * len(head)
  Str  = lambda x   : f"{x:.{decs}f}"     if type(x) is float else str(x)
  say  = lambda w,x : f"{x:>{w}.{decs}f}" if type(x) is float else f"{x:>{w}}"
  says = lambda row : ’ | ’.join([say(w1, x) for w1, x in zip(w, row)])
  for row in [head]+rows:
    w = [max(b4, len(Str(x))) for b4,x in zip(w,row)]
  print(says(head))
  print(’ | ’.join(’-’*(w1) for w1 in w))
  for row in rows: print(says(row))

def shuffle(a):
  random.shuffle(a)
  return a


### Start-up ------------------------------------------------------------------

def main():
  cli(the.__dict__)
  for s in sys.argv:
    if fun := globals().get("eg" + s.replace("-", "_")):
      random.seed(the.rseed)
      fun()

if __name__ == "__main__": main()
```