```python
#!/usr/bin/env python3.8
# vim: ts=2 sw=2 sts=2 et :
import re, sys, copy, math, random
from typing import Any
#  _ _ _    
# | (_) |__ 
# | | | '_ \
# |_|_|_.__/

big = sys.maxsize

is_klass = lambda x: "!" in x
is_less  = lambda x: x[-1] =="-"
is_skip  = lambda x: x[-1] == ":"
is_num   = lambda x: x[0].isupper()
is_goal  = lambda x: "+" in x or "-" in x or is_klass(x)

def atom(x):
  try:    return float(x)
  except: return x

def rows(f):
  with open(f) as fp:
    for line in fp:
      line = re.sub(r'([\n\t\r"\' ]|#.*)', '', line)
      if line: yield [atom(cell.strip()) for cell in line.split(",")]

class o(object):
  def __init__(i, **d): i.__dict__.update(**d)
  def __repr__(i):
    pre = i.__class__.__name__ if isinstance(i,o) else ""
    return pre+'{'+(' '.join([f":{k} {v}" for k, v in
                          sorted(i.__dict__.items()) if str(k)[0] != "_"]))+"}"

def merge(b4:list) -> list:
  j,n,now = -1,len(b4),[]
  while j < n-1:
    j += 1
    a  = b4[j]
    if j < n-2:
      if merged := a.merge(b4[j+1]):
        a  = merged
        j += 1
    now += [a]
  return b4 if len(now)==len(b4) else merge(now)
#          _ 
#  __ ___ | |
# / _/ _ \| |
# \__\___/|_|

class Col(o):
  def __init__(i, at=0, txt=""): i.n,i.at,i.txt = 0,at,txt
  def add(i,x)   : return x
  def dist(i,x,y): return 1 if  x=="?" and y=="?" else i.dist1(x,y)
#      _   _      
#  ___| |_(_)_ __ 
# (_-<| / / | '_ \
# /__/|_\_\_| .__/
#           |_|   

class Skip(Col): ...
#  ___ _  _ _ __  
# (_-<| || | '  \ 
# /__/ \_, |_|_|_|
#      |__/       

class Sym(o):
  def __init__(i,**kw): super().__init__(**kw); i.has = {}

  def add(i, x, inc=1):
    if x !="?": i.has[x] = inc + i.has.get(x,0)

  def dist1(i,x,y): return 0 if x==y else 1

  def div(i):
    p = lambda x: x / (1E-31 + i.n)
    return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )

  def merge(i,j):
    k = Sym(at=i.at, txt=i.txt)
    for x,n in i.has.items(): k.add(x,n)
    for x,n in j.has.items(): k.add(x,n)
    return k

  def spans(i,j, _bins, out):
    xys = [(x,"this",n) for x,n in i.has.items()] + [
           (x,"that",n) for x,n in j.has.items()]
    one, last = None,None
    all  = []
    for x,y,n in sorted(xys, key=first):
      if x != last:
        last = x
        one  = Span(i, x,x)
        all += [one]
      one.add(x,y,n)
    if len(all) > 1 : out += all
#  _ _ _  _ _ __  
# | ' \ || | '  \ 
# |_||_\_,_|_|_|_|

class Num(o):
  def __init__(i,**kw):
    super().__init__(**kw)
    i.w, i.lo, i.hi =  1, big, -big
    if i.txt and is_less(i.txt): i.w = -1

  def add(i,x):
    if x !="?": i.hi = max(i.hi,x);  i.lo = min(i.lo,x)

  def dist(i,x,y):
    if   x=="?" and y=="?": return 1
    if   x=="?" : y = i.norm(y); x = 1 if y < .5 else 0
    elif y=="?" : x = i.norm(x); y = 1 if x < .5 else 0
    else        : x,y = i.norm(x), i.norm(y)
    return abs(x - y)

  def norm(i,x): return 0 if (i.hi-i.lo) < 1E-9 else (x - i.lo) / (i.hi-i.lo)

  def spans(i,j, bins, out):
    lo  = min(i.lo, j.lo)
    hi  = max(i.hi, j.hi)
    gap = (hi-lo) / bins
    xys = [(x,"this",1) for x in i._all] + [
           (x,"that",1) for x in j._all]
    one = Span(i,lo,lo)
    all = [one]
    for x,y,n in sorted(xys, key=first):
      if one.hi - one.lo > gap:
        one  = Span(i, one.hi,x)
        all += [one]
      one.add(x,y,n)
    all = merge(all)
    all[ 0].lo = -big
    all[-1].hi =  big
    if len(all) > 1: out += all
#  ____ __  __ _ _ _  
# (_-< '_ \/ _` | ' \ 
# /__/ .__/\__,_|_||_|
#     |_|             

class Span(o):
  def __init__(i,col, lo, hi, ys=None,):
    i.col, i.lo, i.hi, i.ys = col, lo, hi,  ys or Sym()

  def add(i, x:float, y:Any, inc=1) -> None:
    i.lo = min(x, i.lo)
    i.hi = max(x, i.hi)
    i.ys.add(y,inc)

  def merge(i, j): # -> Span|None
    a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
    if (i.ys.n==0 or j.ys.n==0 or
        c.div()*.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
      return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)

  def selects(i,row:list) -> bool:
    x = row[i.col.at]; return x=="?" or i.lo<=x and x<i.hi

  def show(i)-> None:
    txt = i.col.txt
    if   i.lo == i.hi: return f"{txt} == {i.lo}"
    elif i.lo == -big: return f"{txt} < {i.hi}"
    elif i.hi ==  big: return f"{txt} >= {i.lo}"
    else             : return f"{i.lo} <= {txt} < {i.hi}"

  def support(i) -> float:
    return i.ys.n / i.col.n

  @staticmethod
  def sort(spans : list) -> list:
    "Good spans have large support and low diversity."
    divs, supports = Num(), Num()
    sn = lambda s: supports.norm( s.support())
    dn = lambda s: divs.norm(     s.ys.div())
    f  = lambda s: ((1 - sn(s))**2 + dn(s)**2)**.5/2**.5
    for s in spans:
      divs.add(    s.ys.div())
      supports.add(s.support())
    return sorted(spans, key=f)
#          _    
#  __ ___ | |___
# / _/ _ \| (_-<
# \__\___/|_/__/

class Cols(o):
  def __init__(i,names):
    i.x,i.y = [],[]
    i.all = [i.head(at,txt) for at,txt in enumerate(names)]

  def add(i,lst):
    [col.add(x) for col,x in zip(i.all, lst)]
    return lst

  def head(i,at,txt):
    if is_skip(txt): return Skip(at=at, txt=txt)
    now = (Num if is_num(txt) else Sym)(at=at, txt=txt)
    if is_klass(txt): i.klass=now
    (i.y if is_goal(txt) else i.x).append(now)
    return now
#                          _      
#  ___ __ _ _ __  _ __ | |___ 
# (_-</ _` | '  \| '_ \| / -_)
# /__/\__,_|_|_|_| .__/|_\___|
#                |_|           

class Sample(o):
  def __init__(i,the,inits=None):
    i.rows, i.cols, i.cache = [], None, []
    i.the = copy.deepcopy(the)
    i.lefts, i.rights, i.left, i.right, c = [],[],None,None,0
    if type(inits) == list: [i.add(x) for x in inits]
    if type(inits) == str : [i.add(x) for x in rows(inits)]

  def add(i, sample, row):
    if not i.left:    i.left = row
    elif not i.right: i.right = row
    if i.right and i.left:
      d1 = sample.dist(row, i.left)
      d2 = sample.dist(row, i.right)
      if d1 < d2:
        if d2 > c: i.left, d1,i.c = row,0,d2
      else:
        if d1 > c: i.right,d2,i.c = row,0,d1
      (left if d1 < d2 else right).append(row)

  def clone(i, sample, inits=[]):
    now = i.clone(the)
    now.add(sample, [col.txt for col in i.cols])
    [now.add(sample, row) for row in inits]
    return now

  def dist(i,j,k):
    cols, p = i.cols.x, i.the.p
    d = sum(col.dist(j[col.at], k[col.at])**p for col in cols)
    return (d/len(cols)) ** (1/p)

#---------------------------------------------------
s=Sample(o(p=2),"data/auto93.csv")
```