```
from __future__ import division,print_function
import sys,random,re
sys.dont_write_bytecode =True

################################################
def genic0(**d): return o(
  k=16,
  era=5000,
  num='$',
  klass='=',
    seed=113).update(**d)

def rows0(**d): return o(
  skip="?",
  sep  = ',',
  bad = r'([\"\' \t\r\n]|#.*)'
  ).update(**d)

################################################
rand= random.random
seed= random.seed

def say(c):
  sys.stdout.write(str(c))

def g(lst,n=3):
  for col,val in enumerate(lst):
    if isinstance(val,float):
      val = round(val,n)
    lst[col] = val
  return lst

class o:
  "Define a bag of names slots with no methods."
  def __init__(i,**d): i.update(**d)
  def update(i,**d):
    i.__dict__.update(**d); return i
  def __repr__(i)   :
    d    = i.__dict__
    show = [':%s %s' % (k,d[k])
            for k in sorted(d.keys() )
            if k[0] is not "_"]
    return '{'+' '.join(show)+'}'

################################################
def rows(file,w=None):
  """Leaps over any columns marked 'skip'.
 Turn strings to numbers or strings.
 Kill comments. Join lines that end in 'sep'."""
  w = w or rows0()
  def atom(x):
    try : return int(x)
    except ValueError:
       try : return float(x)
       except ValueError : return x
  def lines():
    n,kept = 0,""
    for line in open(file):
      now   = re.sub(w.bad,"",line)
      kept += now
      if kept:
        if not now[-1] == w.sep:
          yield n, map(atom, kept.split(w.sep))
          n += 1
          kept = ""
  todo = None
  for n,line in lines():
    todo = todo or [col for col,name
                    in enumerate(line)
                    if not w.skip in name]
    yield n, [ line[col] for col in todo ]
```



```
def header(w,row):
  def numOrSym(val):
    return w.num if w.opt.num in val else w.sym
  def indepOrDep(val):
    return w.dep if w.opt.klass in val else w.indep
  for col,val in enumerate(row):
    numOrSym(val).append(col)
    indepOrDep(val).append(col)
    w.name[col] = val
    w.index[val] = col

def data(w,row):
  for col in w.num:
    val = row[col]
    w.min[col] = min(val, w.min.get(col,val))
    w.max[col] = max(val, w.max.get(col,val))

def indep(w,cols):
  for col in cols:
    if col in w.indep: yield col

################################################
def nearest(w,row):
  def norm(val,col):
    lo, hi = w.min[col], w.max[col]
    return (val - lo ) / (hi - lo + 0.00001)
  def dist(centroid):
    n,d = 0,0
    for col in indep(w, w.num):
      x1,x2 = row[col], centroid[col]
      n1,n2 = norm(x1,col), norm(x2,col)
      d    += (n1 - n2)**2
      n    += 1
    for col in indep(w, w.sym):
      x1,x2 = row[col],centroid[col]
      d    += (0 if x1 == x2 else 1)
      n    += 1
    return d**0.5 / n**0.5
  lo, out = 10**32, None
  for n,(_,centroid) in enumerate(w.centroids):
    d = dist(centroid)
    if d < lo:
      lo,out = d,n
  return out

def move(w,new,n):
  u0,old = w.centroids[n]
  u1 = 1
  out = [None]*len(old)
  for col in w.sym:
    x0,x1 = old[col], new[col]
    out[col] = x1 if rand() < 1/(u0+u1) else x0
  for col in w.num:
    x0,x1= old[col], new[col]
    out[col] = (u0*x0 + u1*x1)/ (u0+u1)
  w.centroids[n] = (u0 + u1, out)

def less(w) :
  b4 = len(w.centroids)
  rare = w.opt.era/w.opt.k
  w.centroids = [(1,row) for u,row in
                 w.centroids if u < rare]
  now=len(w.centroids)
  print("-",b4 - now)
```



```
def genic(src='data/diabetes.csv',opt=None):
  w = o(num=[], sym=[], dep=[], indep=[],
        centroids=[],
        min={}, max={}, name={},index={},
        opt=None or genic0())
  for n,row in rows(src):
    if n == 0:
      header(w,row)
    else:
      data(w,row)
      if len(w.centroids) < w.opt.k:
        say("+")
        w.centroids += [(1,row)]
        continue
      move(w,row,nearest(w,row))
      if 0 == (n % w.opt.era):
        say(n)
        less(w)
  return sorted(w.centroids,reverse=True)

if __name__ == '__main__':
  src='data/diabetes2.csv'
  if len(sys.argv) == 2:
    src= sys.argv[1]
  opt=genic0()
  clusters = genic(src)
  seed(opt.seed)
  print("")
  for m,(n,centroid) in enumerate(clusters):
    rare = opt.era/opt.k
    if n > rare:
      print(m+1,n,":",g(centroid,2))
```