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

def cached(f=None,cache={}):
  """To know the active options, cache their
  most recent setting."""
  if f:
    def wrapper(**d):
      tmp = cache[f.__name__] = f(**d)
      return tmp
    return wrapper
  else:
    for x in cache: print(x,cache[x])

#####################################################
@cached
def genic0(**d):
  def tiny(u,w):
    return u <  w.opt.era/w.opt.k/2
  return o(
    k=10,
    era=1000,
    tiny= tiny,
    num='$',
    klass='=',
    seed=1).update(**d)

@cached
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 fun(x):
  return x.__class__.__name__ ≡ 'function'

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

def printm(matrix):
  s = [[str(e) for e in row] for row in matrix]
  lens = [max(map(len, col)) for col in zip(*s)]
  fmt = '\t'.join('{{:{}}}'.format(x) for x in lens)
  for row in [fmt.format(*row) for row in s]:
    print(row)

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)   :
    def name(x):
      return x.__name__ if fun(x) else x
    d    = i.__dict__
    show = [':%s=%s' % (k,name(d[k]))
            for k in sorted(d.keys() )
            if k[0] is ¬ "_"]
    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 ∨ 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 ¬ now[-1] ≡ w.sep:
          yield n, map(atom, kept.split(w.sep))
          n += 1
          kept = ""
  todo = None
  for n,line in lines():
    todo = todo ∨ [col for col,name
                   in enumerate(line)
                   if ¬ 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,u,age,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,u+u1, age, out)

def more(w,n,row):
  w.centroids += [(1,1,n,row)]

def less(w,n) :
  b4 = len(w.centroids)
  w.centroids = [(1,u,dob,row)
                 for u0,u,dob,row in
                 w.centroids
                 if ¬ w.opt.tiny(u0,w)]
  print("n=%s deaths=%s%%" % (
        n,  int(100*(b4 - len(w.centroids))/b4)))

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

def report(w,clusters):
  cols = w.index.keys()
  header = sorted(w.name.keys())
  header= [w.name[i] for i in header]
  matrix = [['gen','caughtLast','caughtAll','dob'] + head
er]
  caught=0
  for m,(u0,u,age,centroid) in enumerate(clusters):
    if ¬ w.opt.tiny(u0,w):
      caught += u0
      matrix += [[m+1,u0,u,age] + g(centroid,2)]
  print("\ncaught in last gen =%s%%\n" %
        int(100*caught/w.opt.era))
  printm(matrix)

if __name__ ≡ '__main__':
  src='data/diabetes.csv'
  if len(sys.argv) ≡ 2:
    src= sys.argv[1]
  print("")
  opt=genic0(era=100,k=8)
  seed(opt.seed)
  report(*genic(src,opt))
  cached()
```