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

def cached(f=None,cache={}):
  "To keep the options, cache their last setting."
  if ¬ f:
    return cache
  def wrapper(**d):
    tmp = cache[f.__name__] = f(**d)
    return tmp
  return wrapper

@cached
def genic0(**d):
  def halfEraDivK(w):
    return w.opt.era/w.opt.k/20
  return o(
    k=10,
    era=67,
    buffer= 250,
    tiny= halfEraDivK,
    num='$',
    klass='=',
    seed=1).update(**d)

@cached
def rows0(**d): return o(
  skip="?",
  sep  = ',',
  bad = r'(["\' \t\r\n]|#.*)',
  zip='data/data.zip' #if None, read text files
  ).update(**d)

rand= random.random
seed= random.seed

def shuffle(lst): random.shuffle(lst); return lst

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 = '|'.join('{{:{}}}'.format(x) for x in lens)
  for row in [fmt.format(*row) for row in s]:
    print(row)

class o:
  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)+'}'

class Col:
  def __init__(i,tag='',col=None):
    i.tag,i.col,i.n = tag,col,1
    i.setup()
  def __iadd__(i,x):
    if x ≠ "?":
      i.n += 1
      i.add(x)
    return i

class S(Col):
  def setup(i): i.cnt,i.most,i.mode = {},0,None
  def xpect(i): return i.mode
  def norm(i,x): return x
  def str2col(x): return x
  def add(i,x):
    tmp  = i.cnt[x] = i.cnt.get(x,0) + 1
```

```
    if tmp > i.most:
      i.most, i.mode = tmp,x

class N(Col):
  def xpect(i): return i.mu
  def str2col(x): return float(x)
  def setup(i):
    i.mu = i.m2 = 0
    i.lo,i.hi = 10**32,−1*10**32
  def add(i,x):
    i.lo, i.hi = min(i.lo,x), max(i.hi,x)
    delta = x − i.mu
    i.mu += delta/i.n
    i.m2 += delta*(x − i.mu)
  def sd(i)  :
    if i.n < 2: return 0
     else:
      return (max(0,i.m2)/(i.n − 1))**0.5
  def norm(i,x):
    tmp = (x − x.lo)/ (x.hi − x.lo + 0.00001)
    return max(0,min(1,tmp))

def content(pattern='*',filezip=None):
  if filezip:
    with zipfile.ZipFile(filezip,'r') as ark:
      for file in ark.namelist():
        if fnmatch.fnmatch(file, pattern):
          with ark.open(file,'r') as lines:
            for line in lines:
              yield file,line
  else:
    for line in open(pattern,'r'):
      yield pattern,line

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 table(file,w):
  def chunks():
    chunk = []
    for m,row in rows(file):
      if m≡0:
        header(w,row)
      else:
        chunk += [row]
        if len(chunk) > w.opt.buffer:
          yield chunk
          chunk=[]
    if chunk: yield chunk
  n=0
  for chunk in chunks():
    for row in shuffle(chunk):
      n += 1
      data(w,row)
      yield n,row

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

def rows(src, w=None):
  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 content(src,zip=w.zip):
      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 fuse(w,new,n):
  u0,u,dob,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, dob, 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 u0 > w.opt.tiny(w)]
  print("at n=%s, pruning %s%% of clusters" % (
        n,  int(100*(b4 − len(w.centroids))/b4)))

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 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'] + header]
  caught=0
  print(len(clusters))
  for m,(u0,u,dob,centroid) in enumerate(clusters):
    print(u0)
    if u0 > w.opt.tiny(w):
      caught += u0
      matrix += [[m+1,u0,u,dob] + g(centroid,2)]
  print("\ncaught in last gen =%s%%\n" %
        int(100*caught/w.opt.era))
  printm(matrix)
  options = cached()
  for x in options: print(x,options[x])
  print("")

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 table(src,w):
    data(w,row)
```

```
      if len(w.centroids) < w.opt.k:
        more(w,n,row)
      else:
        fuse(w,row,nearest(w,row))
        if ¬ (n % w.opt.era):
          less(w,n)
  return w,sorted(w.centroids,reverse=True)

def _genic(src='diabetes.csv'):
  if len(sys.argv) ≡ 2:
    src= sys.argv[1]
  print(src)
  opt=genic0(k=8,era=67)
  seed(opt.seed)
  report(*genic(src,opt))

if __name__ ≡ '__main__': _genic()

"""
data/diabetes2.csv (1.5M records).
caught in last gen =77%

gen | caughtLast | caughtAll | dob     | $preg | $plas  | $pres | $skin | $insu  | $mass | $pedi | $
age  | =class
1   | 205       | 390       | 1571001 | 2.04  | 97.08  | 65.03 | 23.25 | 52.6   | 29.19 | 0.35  | 24.14
| testednegative
2   | 146       | 2408      | 1560001 | 3.77  | 117.73 | 74.08 | 0.79  | 3.86   | 31.04 | 0.4   | 31.84
| testedpositive
3   | 119       | 824       | 1566001 | 7.54  | 142.17 | 78.47 | 7.53  | 16.58  | 29.72 | 0.46  | 52.1
| testednegative
4   | 109       | 252       | 1571002 | 2.39  | 145.63 | 73.09 | 30.13 | 201.47 | 34.58 | 0.35  | 28.5
7 | testednegative
5   | 106       | 2690      | 1554001 | 8.03  | 106.6  | 76.56 | 32.07 | 64.18  | 34.63 | 0.41  | 40.8
4 | testednegative
6   | 85        | 654       | 1569002 | 1.62  | 118.5  | 70.76 | 33.44 | 119.23 | 36.16 | 0.93  | 26.23
 | testedpositive
genic0 {:buffer=500 :era=1000 :k=8 :klass== :num=$ :seed=1 :tiny=halfEraDivK}
rows0 {:bad=(["\' \t\r\n]|#.*) :sep=, :skip=?}

real        3m25.949s
user        3m7.403s
sys         0m2.315s
"""
```