```
#Structure from SA Lecture
import sys,re,random,math
sys.dont_write_bytecode = True

from options import *
from utils import *
from sk import *

myOpt = Options()

class Analyzer:
  n = 50
  old = [1 for i in range (0, n)]
  new = [1 for i in range (0, n)]
  era_lives = myOpt.era_lives;

  def bettered(self, new, old):

    def quartiles(value):
      return value*.25, value*.5, value*.75

    def betterifless():
      p1, median1, p3 = quartiles(new)
      IQR1=p3−p1
      p1, median2, p3 = quartiles(old)
      IQR2=p3−p1
      return median1<median2, IQR1<IQR2

    def same(): return a12(new, old)≤0.56

    betterMedian, betterIQR = betterifless()
    return betterMedian, betterIQR, same()

  def EraStop(self, lst):
    self.old = self.new
    self.new = lst
    out = False
    betterMedian = False
    betterIQR = False
    same = False

    #print self.old
    #print self.new
    oldQ1, oldMedian, oldQ3 = quartiles(self.old)
    newQ1, newMedian, newQ3 = quartiles(self.new)
    if newMedian < oldMedian:
      betterMedian = True
    if newQ3 − newQ1 < oldQ3 − oldQ1:
      betterIQR = True
    if a12(self.new, self.old) ≤ myOpt.a12_test:
      same = True

    #worsed
    if (same ∧ ¬ betterIQR) ∨ (¬ same ∧ ¬ betterMedian):
      out = False
    #bettered
    elif (¬ same ∧ betterMedian):
      out = True

    if out:
      self.era_lives += 1
    else:
      self.era_lives −= 1
    if self.era_lives ≡ 0:
      #print "Early Era Termination!"
      return True
    else:
      return False

#from menzies code
def median(lst,ordered=False):
  if ¬ ordered: lst= sorted(lst)
  n = len(lst)
  p = n//2
  if n % 2: return lst[p]
  q = p − 1
  q = max(0,min(q,n))
  return (lst[p] + lst[q])/2

def quartiles(lst):
  q1 = lst[int(len(lst)*.25)]
  med = median(lst, False)
  q3 = lst[int(len(lst)*.5)]
  return q1, med, q3
```

```
Baseline:  2.10527662818 ,  6.32475427202
(  ---  *   -|----        ), 0.11728,  0.22055,  0.29839,  0.46375,  0.69740
(- *-        |            ), 0.03786,  0.05512,  0.11345,  0.15036,  0.17421
(-  *        |            ), 0.00143,  0.06111,  0.13602,  0.13757,  0.19635
===============================================================
Model Name:  ZDT3
Searcher Name:  MWS
Seed: 1 Lives:  3
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e-05 Slices: 10
Time to run (s):  0.32783
Runs:  1
Average per run (s):  0.32783
(*           |            ), -0.00308,  -0.00308,  -0.00308,  -0.00308,  -0.00308
===============================================================


Baseline:  0.239532682762 ,  4541.57323683
(*           |            ), 0.00498,  0.00498,  0.00498,  0.00558,  0.00699
(*           |            ), 0.00403,  0.00473,  0.00498,  0.00498,  0.00498
(*           |            ), 0.00411,  0.00498,  0.00498,  0.00498,  0.01144
(*           |            ), 0.00390,  0.00498,  0.00498,  0.00498,  0.00498
(*           |            ), 0.00403,  0.00476,  0.00498,  0.00498,  0.00822
(*           |            ), 0.00403,  0.00498,  0.00498,  0.00498,  0.00527
(*           |            ), 0.00498,  0.00498,  0.00498,  0.00553,  0.00822
(*           |            ), 0.00498,  0.00498,  0.00498,  0.00498,  0.00755
(*           |            ), 0.00420,  0.00498,  0.00498,  0.01130,  0.01422
(*           |            ), 0.00398,  0.00498,  0.00498,  0.00498,  0.00628
(*           |            ), 0.00497,  0.00498,  0.00498,  0.00514,  0.00945
(*           |            ), 0.00463,  0.00498,  0.00498,  0.00498,  0.00782
No Best Found for prob =  0.75
===============================================================
Model Name:  Viennet3
Searcher Name:  MWS
Seed: 1 Lives:  3
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e-05 Slices: 10
Time to run (s):  0.058218
No valid runs!
===============================================================
rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x1 ,      51  ,    11 (*              |              ), 0.34,  0.49,  0.51,  0.51,  0.60
   2 ,       x2 ,     800  ,   200 (               |   ----   *-- ), 6.00,  7.00,  8.00,  8.00,  9.00

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x1 ,      30  ,    20 (*              |              ), 0.10,  0.20,  0.30,  0.30,  0.40
   1 ,       x2 ,      30  ,    20 (*              |              ), 0.10,  0.20,  0.30,  0.30,  0.40
   2 ,       x3 ,     800  ,   200 (               |   ----   *-- ), 6.00,  7.00,  8.00,  8.00,  9.00

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x5 ,      30  ,    20 (---    *---    |              ), 0.10,  0.20,  0.30,  0.30,  0.40
   1 ,       x3 ,      35  ,    15 ( ----    *-    |              ), 0.15,  0.25,  0.35,  0.35,  0.40
   2 ,       x1 ,      51  ,    11 (        ------ *--            ), 0.34,  0.49,  0.51,  0.51,  0.60
   3 ,       x2 ,      80  ,    20 (               |  ----    *-- ), 0.60,  0.70,  0.80,  0.80,  0.90
   3 ,       x4 ,      80  ,    20 (               |  ----    *-- ), 0.60,  0.70,  0.80,  0.80,  0.90

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x1 ,    10000  ,   150 (-------       *|              ),99.00, 99.50, 100.00, 101.00, 101.00
   1 ,       x2 ,    10000  ,   100 (--------------*|              ),99.00, 100.00, 100.00, 101.00, 101.00
   1 ,       x3 ,    10000  ,   150 (-------       *|              ),99.00, 99.50, 100.00, 101.00, 101.00
   1 ,       x4 ,    10000  ,   100 (--------------*|              ),99.00, 100.00, 100.00, 101.00, 101.00

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x1 ,    1100  ,     0 ( *             |              ),11.00, 11.00, 12.00, 13.00, 13.00
   1 ,       x2 ,    1400  ,     0 (              *|              ),14.00, 14.00, 22.00, 31.00, 31.00
   2 ,       x3 ,    2300  ,     0 (               |*             ),23.00, 23.00, 24.00, 31.00, 31.00
   2 ,       x5 ,    3200  ,     0 (               |            * ),32.00, 32.00, 33.00, 34.00, 34.00

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x1 ,    1100  ,     0 (*              |              ),11.00, 11.00, 11.00, 11.00, 11.00
   1 ,       x2 ,    1100  ,     0 (*              |              ),11.00, 11.00, 11.00, 11.00, 11.00
   1 ,       x3 ,    1100  ,     0 (*              |              ),11.00, 11.00, 11.00, 11.00, 11.00

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x1 ,    1100  ,     0 (*              |              ),11.00, 11.00, 11.00, 11.00, 11.00
   1 ,       x2 ,    1100  ,     0 (*              |              ),11.00, 11.00, 11.00, 11.00, 11.00
   2 ,       x4 ,    3400  ,   200 (               |          - * ),32.00, 33.00, 34.00, 34.00, 35.00

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,       x2 ,      24  ,    54 (--     *       |---------     ), 0.01,  0.10,  0.24,  0.55,  0.86
   2 ,       x3 ,      51  ,    49 (   ------      *    ------    ), 0.10,  0.30,  0.51,  0.67,  0.88
   3 ,       x1 ,      71  ,    38 (        -------|     *  ----  ), 0.28,  0.53,  0.71,  0.83,  0.94

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,     kids ,    2000  ,   300 ( - *           |              ),16.00, 18.00, 20.00, 21.00, 23.00
   2 ,     cook ,    3200  ,  3000 (  --   *      -|-------       ),18.00, 23.00, 32.00, 53.00, 78.00
   2 ,   novels ,    4400  ,  2000 (     --    *  -|----          ),28.00, 33.00, 44.00, 53.00, 71.00
   2 ,    zines ,    6300  ,  3900 (-----------    | *      ----- ),11.00, 43.00, 63.00, 82.00, 98.00
```

```
===============================================================
Model Name:  ZDT3
Searcher Name:  SA
Seed: 1 Lives:  3
SA Options:
KMAX: 500 Cooling: 0.6
Time to run (s):  1.311633
Runs:  30
Average per run (s):  0.0437211
(*           |            ), 0.03007,  0.12892,  0.18899,  0.23969,  0.36037
===============================================================
===============================================================
Model Name:  ZDT3
Searcher Name:  MWS
Seed: 1 Lives:  3
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e−05 Slices: 10
Time to run (s):  1.91517
Runs:  30
Average per run (s):  0.063839
(*           |            ), −0.03178,  −0.01606,  −0.00944,  0.01218,  0.21089
===============================================================
Scott−Knott for ZDT3

rank ,     name ,    med   ,  iqr
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   1 ,     MWS ,      −1  ,     3 (       −*−−−−−−|−−            ),−0.03, −0.02, −0.01,  0.01,  0.21
   2 ,      SA ,      18  ,    19 (          −−−− | * −−−−−      ), 0.03,  0.13,  0.19,  0.24,  0.36


===============================================================
Model Name:  Viennet3
Searcher Name:  SA
Seed: 1 Lives:  3
SA Options:
KMAX: 500 Cooling: 0.6
Time to run (s):  0.20987
Runs:  30
Average per run (s):  0.00699566666667
(*           |            ), −0.00020,  0.00538,  0.01491,  0.03240,  0.05654
===============================================================
===============================================================
Model Name:  Viennet3
Searcher Name:  MWS
Seed: 1 Lives:  3
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e−05 Slices: 10
Time to run (s):  0.290776
Runs:  30
Average per run (s):  0.00969253333333
(*           |            ), 0.01141,  0.03698,  0.07352,  0.12363,  0.20950
===============================================================
Scott−Knott for Viennet3

rank ,     name ,    med   ,  iqr
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
   1 ,      SA ,       1  ,     3 (   − *−−−      |             ),−0.00,  0.01,  0.01,  0.03,  0.06
   2 ,     MWS ,       7  ,    11 (    −−−    *   |−−−−−−−−      ), 0.01,  0.04,  0.07,  0.12,  0.21
```

```
===============================================================
Model Name:  ZDT1
Searcher Name:  SA
Seed: 1 Lives:  3
SA Options:
KMAX: 500 Cooling: 0.6
Time to run (s):  1.803805
Runs:  5
Average per run (s):  0.360761
(*           |            ), 0.07102,  0.11667,  0.21154,  0.22530,  0.24614
===============================================================
===============================================================
Model Name:  ZDT1
Searcher Name:  MWS
Seed: 1 Lives:  3
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e-05 Slices: 10
Time to run (s):  1.882665
Runs:  5
Average per run (s):  0.376533
(*           |            ), -0.05222,  -0.03011,  -0.02254,  -0.00696,  -0.00374
===============================================================
Scott-Knott for ZDT1

rank ,     name ,    med   ,  iqr
----------------------------------------------------
   1 ,      MWS ,      -2  ,     2 (--*            |              ),-0.05, -0.03, -0.02, -0.01, -0.00
   2 ,       SA ,      21  ,    11 (            ---|          *-- ), 0.07,  0.12,  0.21,  0.23,  0.25
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

class Options:
  #Globals
  debug = False
  seed = 1
  era_lives = 3
  a12_test = 0.6

  #MaxWalkSat options
  mws_prob = 0.75
  mws_maxTries = 500
  mws_maxChanges = 500
  mws_threshold = .00001
  mws_slices = 10

  #Simulated Annealing options
  sa_kmax = 500
  sa_cooling = .6

  def printGlobals(self):
    print "Seed:", self.seed, "Lives: ", self.era_lives
```

```
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from sk import *

def rdiv8():
  rdivDemo([
      ["novels", 28, 33, 44, 71, 53],
      ["kids", 23, 18, 16, 20, 21],
      ["zines", 11, 98, 43, 63, 82],
      ["cook", 23, 18, 32, 53, 78],
      ])

rdiv8()
```

```python
import sys
from datetime import datetime
import random

sys.dont_write_bytecode = True

from models import *
from searchers import *
from utils import *
from options import *
from sk import *

myOpt = Options()

#Inspired by vivekaxl’s display function
def display(model, searcher, startTime, scores, r):
  print "============================================================"
  print "Model Name: ", model.__name__
  print "Searcher Name: ", searcher.__class__.__name__
  diff = (datetime.now() − startTime).total_seconds()
  myOpt.printGlobals()
  searcher.printOptions()
  print "Time to run (s): ", diff
  if r ≡ 0:
    print "No valid runs!"
  else:
    print "Runs: ", r
    print "Average per run (s): ", diff/r
    print xtile(scores,width=25,show=" %1.5f")
  print "============================================================"

def main(modelList, searcherList):
  r = 5
  for klass in modelList:
    classScoreList = []
    for searcher in searcherList:
      fullScoreList = []
      startTime = datetime.now()
      scores = []
      mySearcher = searcher()
      random.seed(myOpt.seed)
      for _ in range(r):
        myKlass = klass()
        result, valid = mySearcher.run(myKlass)
        if valid ≡ True:
          scores.append(result)
      display(klass, mySearcher, startTime, scores, len(scores))
      fullScoreList.append(searcher.__name__)
      for x in scores:
        fullScoreList.append(x)
      classScoreList.append(fullScoreList)
    print "Scott−Knott for", klass.__name__
    rdivDemo(classScoreList)
    print "\n"


#modelList = [ZDT3, Viennet3]
#searcherList = [SA, MWS]
modelList = [ZDT1]
searcherList = [SA, MWS]

main(modelList, searcherList)
```

```
"""

## Hyptotheis Testing Stuff


### Standard Stuff

#### Standard Headers

"""
from __future__ import division
import sys,random,math
sys.dont_write_bytecode = True
"""

#### Standard Utils

"""
class o():
  "Anonymous container"
  def __init__(i,**fields) :
    i.override(fields)
  def override(i,d): i.__dict__.update(d); return i
  def __repr__(i):
    d = i.__dict__
    name = i.__class__.__name__
    return name+'{'+' '.join([':%s %s' % (k,pretty(d[k]))
                     for k in i.show()])+ '}'
  def show(i):
    return [k for k in sorted(i.__dict__.keys())
            if ¬ "_" in k]
"""

Misc functions:

"""
rand = random.random
any  = random.choice
seed = random.seed
exp  = lambda n: math.e**n
ln   = lambda n: math.log(n,math.e)
g    = lambda n: round(n,2)

def median(lst,ordered=False):
  if ¬ ordered: lst= sorted(lst)
  n = len(lst)
  p = n//2
  if n % 2: return lst[p]
  q = p − 1
  q = max(0,min(q,n))
  return (lst[p] + lst[q])/2

def msecs(f):
  import time
  t1 = time.time()
  f()
  return (time.time() − t1) * 1000

def pairs(lst):
  "Return all pairs of items i,i+1 from a list."
  last=lst[0]
  for i in lst[1:]:
    yield last,i
    last = i

def xtile(lst,lo=0,hi=100,width=50,
             chops=[0.1 ,0.3,0.5,0.7,0.9],
             marks=["−" ," "," ","−"," "],
             bar="|",star="*",show=" %3.0f"):
  """The function _xtile_ takes a list of (possibly)
 unsorted numbers and presents them as a horizontal
 xtile chart (in ascii format). The default is a
 contracted _quintile_ that shows the
 10,30,50,70,90 breaks in the data (but this can be
 changed− see the optional flags of the function).
 """
  def pos(p)   : return ordered[int(len(lst)*p)]
  def place(x) :
    return int(width*float((x − lo))/(hi − lo+0.00001))
  def pretty(lst) :
    return ','.join([show % x for x in lst])
  ordered = sorted(lst)
  lo      = min(lo,ordered[0])
  hi      = max(hi,ordered[−1])
  what    = [pos(p)   for p in chops]
  where   = [place(n) for n in  what]
  out     = [" "] * width
  for one,two in pairs(where):
    for i in range(one,two):
      out[i] = marks[0]
    marks = marks[1:]
  out[int(width/2)]    = bar
  out[place(pos(0.5))] = star
  return '('+''.join(out) +  ")," +  pretty(what)
```

```
def _tileX() :
  import random
  random.seed(1)
  nums = [random.random()**2 for _ in range(100)]
  print xtile(nums,lo=0,hi=1.0,width=25,show=" %5.2f")
"""

### Standard Accumulator for Numbers

Note the _lt_ method: this accumulator can be sorted by median values.

Warning: this accumulator keeps _all_ numbers. Might be better to use
a bounded cache.

"""
class Num:
  "An Accumulator for numbers"
  def __init__(i,name,inits=[]):
    i.n = i.m2 = i.mu = 0.0
    i.all=[]
    i._median=None
    i.name = name
    i.rank = 0
    for x in inits: i.add(x)
  def s(i)       : return (i.m2/(i.n − 1))**0.5
  def add(i,x):
    i._median=None
    i.n   += 1
    i.all += [x]
    delta  = x − i.mu
    i.mu  += delta*1.0/i.n
    i.m2  += delta*(x − i.mu)
  def __add__(i,j):
    return Num(i.name + j.name,i.all + j.all)
  def quartiles(i):
    def p(x) : return int(100*g(xs[x]))
    i.median()
    xs = i.all
    n  = int(len(xs)*0.25)
    return p(n) , p(2*n) , p(3*n)
  def median(i):
    if ¬ i._median:
      i.all = sorted(i.all)
      i._median=median(i.all)
    return i._median
  def __lt__(i,j):
    return i.median() < j.median()
  def spread(i):
    i.all=sorted(i.all)
    n1=i.n*0.25
    n2=i.n*0.75
    if len(i.all) ≤ 1:
      return 0
    if len(i.all) ≡ 2:
      return i.all[1] − i.all[0]
    else:
      return i.all[int(n2)] − i.all[int(n1)]


"""

### The A12 Effect Size Test

"""
def a12slow(lst1,lst2):
  "how often is x in lst1 more than y in lst2?"
  more = same = 0.0
  for x in lst1:
    for y in lst2:
      if    x ≡ y : same += 1
      elif  x >  y : more += 1
  x= (more + 0.5*same) / (len(lst1)*len(lst2))
  return x

def a12(lst1,lst2):
  "how often is x in lst1 more than y in lst2?"
  def loop(t,t1,t2):
    while t1.j < t1.n ∧ t2.j < t2.n:
      h1 = t1.l[t1.j]
      h2 = t2.l[t2.j]
      h3 = t2.l[t2.j+1] if t2.j+1 < t2.n else None
      if h1>  h2:
        t1.j  += 1; t1.gt += t2.n − t2.j
      elif h1 ≡ h2:
        if h3 ∧ h1 > h3 :
            t1.gt += t2.n − t2.j  − 1
        t1.j  += 1; t1.eq += 1; t2.eq += 1
      else:
        t2,t1  = t1,t2
    return t.gt*1.0, t.eq*1.0
  #−−−−−−−−−−−−−−−−−−−−−−−−−−
  lst1 = sorted(lst1, reverse=True)
  lst2 = sorted(lst2, reverse=True)
  n1   = len(lst1)
  n2   = len(lst2)
  t1   = o(l=lst1,j=0,eq=0,gt=0,n=n1)
  t2   = o(l=lst2,j=0,eq=0,gt=0,n=n2)
  gt,eq= loop(t1, t1, t2)
  return gt/(n1*n2) + eq/2/(n1*n2)

def _a12():
  def f1(): return a12slow(l1,l2)
  def f2(): return a12(l1,l2)
  for n in [100,200,400,800,1600,3200,6400]:
    l1 = [rand() for _ in xrange(n)]
    l2 = [rand() for _ in xrange(n)]
    t1 = msecs(f1)
    t2 = msecs(f2)
    print n, g(f1()),g(f2()),int((t1/t2))


"""Output:

‘‘‘‘
n   a12(fast)       a12(slow)       tfast / tslow
−−− −−−−−−−−−−−−−−− −−−−−−−−−−−−−−− −−−−−−−−−−−−−−−
100  0.53          0.53             4
200  0.48          0.48             6
400  0.49          0.49            28
800  0.5          0.5             26
1600 0.51          0.51            72
3200 0.49          0.49           109
6400 0.5           0.5            244
‘‘‘‘


## Non−Parametric Hypothesis Testing

The following _bootstrap_ method was introduced in
1979 by Bradley Efron at Stanford University. It
was inspired by earlier work on the
jackknife.
Improved estimates of the variance were [developed later][efron01].

[efron01]: http://goo.gl/14n8Wf "Bradley Efron ∧ R.J. Tibshirani. An Introduction to the Bootstrap (Chapman & Hall/CRC M


To check if two populations _(y0,z0)_
are different, many times sample with replacement
from both to generate _(y1,z1), (y2,z2), (y3,z3)_.. etc.

"""
def sampleWithReplacement(lst):
  "returns a list same size as list"
  def any(n)  : return random.uniform(0,n)
  def one(lst): return lst[ int(any(len(lst))) ]
  return [one(lst) for _ in lst]
"""


Then, for all those samples,
 check if some *testStatistic* in the original pair
hold for all the other pairs. If it does more than (say) 99%
of the time, then we are 99% confident in that the
populations are the same.

In such a _bootstrap_ hypothesis test, the *some property*
is the difference between the two populations, muted by the
joint standard deviation of the populations.

"""
def testStatistic(y,z):
    """Checks if two means are different, tempered
  by the sample size of ’y’ and ’z’"""
    tmp1 = tmp2 = 0
    for y1 in y.all: tmp1 += (y1 − y.mu)**2
    for z1 in z.all: tmp2 += (z1 − z.mu)**2
    s1    = (float(tmp1)/(y.n − 1))**0.5
    s2    = (float(tmp2)/(z.n − 1))**0.5
    delta = z.mu − y.mu
    if s1+s2:
      delta =  delta/((s1/y.n + s2/z.n)**0.5)
    return delta
"""

The rest is just details:

+ Efron advises
  to make the mean of the populations the same (see
  the _yhat,zhat_ stuff shown below).
+ The class _total_ is a just a quick and dirty accumulation class.
+ For more details see [the Efron text][efron01].

"""
def bootstrap(y0,z0,conf=0.01,b=1000):
  """The bootstrap hypothesis test from
  p220 to 223 of Efron’s book ’An
  introduction to the boostrap."""
  class total():
    "quick and dirty data collector"
```

```
    def __init__(i,some=[]):
      i.sum = i.n = i.mu = 0 ; i.all=[]
      for one in some: i.put(one)
    def put(i,x):
      i.all.append(x);
      i.sum +=x; i.n += 1; i.mu = float(i.sum)/i.n
    def __add__(i1,i2): return total(i1.all + i2.all)
  y, z   = total(y0), total(z0)
  x      = y + z
  tobs   = testStatistic(y,z)
  yhat   = [y1 − y.mu + x.mu for y1 in y.all]
  zhat   = [z1 − z.mu + x.mu for z1 in z.all]
  bigger = 0.0
  for i in range(b):
    if testStatistic(total(sampleWithReplacement(yhat)),
                     total(sampleWithReplacement(zhat))) > tobs:
      bigger += 1
  return bigger / b < conf
"""

#### Examples

"""
def _bootstraped():
  def worker(n=1000,
             mu1=10,  sigma1=1,
             mu2=10.2, sigma2=1):
    def g(mu,sigma) : return random.gauss(mu,sigma)
    x = [g(mu1,sigma1) for i in range(n)]
    y = [g(mu2,sigma2) for i in range(n)]
    return n,mu1,sigma1,mu2,sigma2,\
        ’different’ if bootstrap(x,y) else ’same’
  # very different means, same std
  print worker(mu1=10, sigma1=10,
               mu2=100, sigma2=10)
  # similar means and std
  print worker(mu1= 10.1, sigma1=1,
               mu2= 10.2, sigma2=1)
  # slightly different means, same std
  print worker(mu1= 10.1, sigma1= 1,
               mu2= 10.8, sigma2= 1)
  # different in mu eater by large std
  print worker(mu1= 10.1, sigma1= 10,
               mu2= 10.8, sigma2= 1)
"""

Output:

‘‘‘‘
_bootstraped()

(1000, 10, 10, 100, 10, ’different’)
(1000, 10.1, 1, 10.2, 1, ’same’)
(1000, 10.1, 1, 10.8, 1, ’different’)
(1000, 10.1, 10, 10.8, 1, ’same’)
‘‘‘‘

Warning− the above took 8 seconds to generate since we used 1000 bootstraps.
As to how many bootstraps are enough, that depends on the data. There are
results saying 200 to 400 are enough but, since I am  suspicious man, I run it for 1000.

Which means the runtimes associated with bootstrapping is a significant issue.
To reduce that runtime, I avoid things like an all−pairs comparison of all treatments
(see below: Scott−knott).  Also, BEFORE I do the boostrap, I first run
the effect size test (and only go to bootstrapping in effect size passes:

"""
def different(l1,l2):
  #return bootstrap(l1,l2) and a12(l2,l1)
  return a12(l2,l1) ∧ bootstrap(l1,l2)

"""

## Saner Hypothesis Testing

The following code, which you should use verbatim does the following:


+ All treatments are clustered into _ranks_. In practice, dozens
  of treatments end up generating just a handful of ranks.
+ The numbers of calls to the hypothesis tests are minimized:
    + Treatments are sorted by their median value.
    + Treatments are divided into two groups such that the
      expected value of the mean values _after_ the split is minimized;
    + Hypothesis tests are called to test if the two groups are truly difference.
          + All hypothesis tests are non−parametric and include (1) effect size tests
            and (2) tests for statistically significant numbers;
          + Slow bootstraps are executed  if the faster _A12_ tests are passed;

In practice, this means that the hypothesis tests (with confidence of say, 95%)
are called on only a logarithmic number of times. So...

+ With this method, 16 treatments can be studied using less than _&sum;<sub>1,2,4,8,16</sub>log<sub>2</sub>i =15_ hypothesis tests  and confidence _0.99<sup
+ But if did this with the 120 all−pairs comparisons of the 16 treatments, we would have total confidence _0.99<sup>120</sup>=0.30.
```

```
For examples on using this code, see _rdivDemo_ (below).

"""
def scottknott(data,cohen=0.3,small=3, useA12=False,epsilon=0.01):
  """Recursively split data, maximizing delta of
 the expected value of the mean before and
 after the splits.
 Reject splits with under 3 items"""
  all  = reduce(lambda x,y:x+y,data)
  same = lambda l,r: abs(l.median() - r.median()) ≤ all.s()*cohen
  if useA12:
    same = lambda l, r:   ¬ different(l.all,r.all)
  big  = lambda    n: n > small
  return rdiv(data,all,minMu,big,same,epsilon)

def rdiv(data,  # a list of class Nums
         all,   # all the data combined into one num
         div,   # function: find the best split
         big,   # function: rejects small splits
         same, # function: rejects similar splits
         epsilon): # small enough to split two parts
  """Looks for ways to split sorted data,
 Recurses into each split. Assigns a 'rank' number
 to all the leaf splits found in this way.
 """
  def recurse(parts,all,rank=0):
    "Split, then recurse on each part."
    cut,left,right = maybeIgnore(div(parts,all,big,epsilon),
                                 same,parts)
    if cut:
      # if cut, rank "right" higher than "left"
      rank = recurse(parts[:cut],left,rank) + 1
      rank = recurse(parts[cut:],right,rank)
    else:
      # if no cut, then all get same rank
      for part in parts:
        part.rank = rank
    return rank
  recurse(sorted(data),all)
  return data

def maybeIgnore((cut,left,right), same,parts):
  if cut:
    if same(sum(parts[:cut],Num('upto')),
            sum(parts[cut:],Num('above'))):
      cut = left = right = None
  return cut,left,right

def minMu(parts,all,big,epsilon):
  """Find a cut in the parts that maximizes
 the expected value of the difference in
 the mean before and after the cut.
 Reject splits that are insignificantly
 different or that generate very small subsets.
 """
  cut,left,right = None,None,None
  before, mu     =  0, all.mu
  for i,l,r in leftRight(parts,epsilon):
    if big(l.n) ∧ big(r.n):
      n   = all.n * 1.0
      now = l.n/n*(mu- l.mu)**2 + r.n/n*(mu- r.mu)**2
      if now > before:
        before,cut,left,right = now,i,l,r
  return cut,left,right

def leftRight(parts,epsilon=0.01):
  """Iterator. For all items in 'parts',
 return everything to the left and everything
 from here to the end. For reasons of
 efficiency, take a first pass over the data
 to pre-compute and cache right-hand-sides
 """
  rights = {}
  n = j = len(parts) - 1
  while j > 0:
    rights[j] = parts[j]
    if j < n: rights[j] += rights[j+1]
    j -=1
  left = parts[0]
  for i,one in enumerate(parts):
    if i> 0:
      if parts[i]._median - parts[i-1]._median > epsilon:
        yield i,left,rights[i]
      left += one
"""

## Putting it All Together

Driver for the demos:

"""
def rdivDemo(data):
  def z(x):
    return int(100 * (x - lo) / (hi - lo + 0.00001))
  data = map(lambda lst:Num(lst[0],lst[1:]),
```

```
             data)
  print ""
  ranks=[]
  for x in scottknott(data,useA12=True):
    ranks += [(x.rank,x.median(),x)]
  all=[]
  for _,__,x in sorted(ranks): all += x.all
  all = sorted(all)
  lo, hi = all[0], all[-1]
  line = "----------------------------------------------------"
  last = None
  print  ('%4s , %8s ,  %s  , %4s ' % \
               ('rank', 'name', 'med', 'iqr'))+ "\n"+ line
  for _,__,x in sorted(ranks):
    q1,q2,q3 = x.quartiles()
    print  ('%4s , %8s ,  %4s , %4s ' % \
                 (x.rank+1, x.name, q2, q3 - q1))  + \
              xtile(x.all,lo=lo,hi=hi,width=30,show="%5.2f")
    last = x.rank

def rdiv0():
  rdivDemo([
        ["x1",0.34, 0.49, 0.51, 0.6],
        ["x2",6,  7,  8,  9] ])
"""

````
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x1 ,    51  ,   11 (*          |          ), 0.34,  0.49,  0.51,  0.51,  0.60
  2 ,       x2 ,   800  ,  200 (           | ----  *-- ), 6.00,  7.00,  8.00,  8.00,  9.00
````

"""
def rdiv1():
  rdivDemo([
        ["x1",0.1, 0.2, 0.3, 0.4],
        ["x2",0.1, 0.2, 0.3, 0.4],
        ["x3",6, 7, 8, 9]])
"""

````
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x1 ,    30  ,   20 (*          |          ), 0.10,  0.20,  0.30,  0.30,  0.40
  1 ,       x2 ,    30  ,   20 (*          |          ), 0.10,  0.20,  0.30,  0.30,  0.40
  2 ,       x3 ,   800  ,  200 (           | ----  *-- ), 6.00,  7.00,  8.00,  8.00,  9.00
````

"""
def rdiv2():
  rdivDemo([
        ["x1",0.34, 0.49, 0.51, 0.6],
        ["x2",0.6, 0.7, 0.8, 0.9],
        ["x3",0.15, 0.25, 0.4, 0.35],
        ["x4",0.6, 0.7, 0.8, 0.9],
        ["x5",0.1, 0.2, 0.3, 0.4]])
"""

````
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x5 ,    30  ,   20 (---   *---  |          ), 0.10,  0.20,  0.30,  0.30,  0.40
  1 ,       x3 ,    35  ,   15 ( ----   *-  |          ), 0.15,  0.25,  0.35,  0.35,  0.40
  2 ,       x1 ,    51  ,   11 (     ------ *--          ), 0.34,  0.49,  0.51,  0.51,  0.60
  3 ,       x2 ,    80  ,   20 (           | ----  *-- ), 0.60,  0.70,  0.80,  0.80,  0.90
  3 ,       x4 ,    80  ,   20 (           | ----  *-- ), 0.60,  0.70,  0.80,  0.80,  0.90
````

"""
def rdiv3():
  rdivDemo([
      ["x1",101, 100, 99,   101, 99.5],
      ["x2",101, 100, 99,   101, 100],
      ["x3",101, 100, 99.5, 101, 99],
      ["x4",101, 100, 99,   101, 100]])
"""

````
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x1 ,   10000  ,  150 (-------     *|          ),99.00, 99.50, 100.00, 101.00, 101.00
  1 ,       x2 ,   10000  ,  100 (--------------*|          ),99.00, 100.00, 100.00, 101.00, 101.00
  1 ,       x3 ,   10000  ,  150 (-------     *|          ),99.00, 99.50, 100.00, 101.00, 101.00
  1 ,       x4 ,   10000  ,  100 (--------------*|          ),99.00, 100.00, 100.00, 101.00, 101.00
~~~

"""
def rdiv4():
  rdivDemo([
      ["x1",11,12,13],
      ["x2",14,31,22],
      ["x3",23,24,31],
      ["x5",32,33,34]])
"""
```
~~~

~~~
‘‘‘‘
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x1 ,   1100  ,    0 ( *         |          ),11.00, 11.00, 12.00, 13.00, 13.00
  1 ,       x2 ,   1400  ,    0 (          *|          ),14.00, 14.00, 22.00, 31.00, 31.00
  2 ,       x3 ,   2300  ,    0 (           |*         ),23.00, 23.00, 24.00, 31.00, 31.00
  2 ,       x5 ,   3200  ,    0 (           |        * ),32.00, 32.00, 33.00, 34.00, 34.00
‘‘‘‘

"""
def rdiv5():
  rdivDemo([
      ["x1",11,11,11],
      ["x2",11,11,11],
      ["x3",11,11,11]])
"""

‘‘‘‘
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x1 ,   1100  ,    0 (*          |          ),11.00, 11.00, 11.00, 11.00, 11.00
  1 ,       x2 ,   1100  ,    0 (*          |          ),11.00, 11.00, 11.00, 11.00, 11.00
  1 ,       x3 ,   1100  ,    0 (*          |          ),11.00, 11.00, 11.00, 11.00, 11.00
‘‘‘‘

"""
def rdiv6():
  rdivDemo([
      ["x1",11,11,11],
      ["x2",11,11,11],
      ["x4",32,33,34,35]])
"""

‘‘‘‘
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x1 ,   1100  ,    0 (*          |          ),11.00, 11.00, 11.00, 11.00, 11.00
  1 ,       x2 ,   1100  ,    0 (*          |          ),11.00, 11.00, 11.00, 11.00, 11.00
  2 ,       x4 ,   3400  ,  200 (           |       - * ),32.00, 33.00, 34.00, 34.00, 35.00
‘‘‘‘

"""
def rdiv7():
  rdivDemo([
    ["x1"] +  [rand()**0.5 for _ in range(256)],
    ["x2"] +  [rand()**2   for _ in range(256)],
    ["x3"] +  [rand()      for _ in range(256)]
  ])
"""

‘‘‘‘
rank ,      name ,   med   , iqr
----------------------------------------------------
  1 ,       x2 ,    25  ,   50 (--    *    -|----------    ), 0.01,  0.09,  0.25,  0.47,  0.86
  2 ,       x3 ,    49  ,   47 ( ------     *|  --------    ), 0.08,  0.29,  0.49,  0.66,  0.89
  3 ,       x1 ,    73  ,   37 (      ------|-   *  --- ), 0.32,  0.57,  0.73,  0.86,  0.95
‘‘‘‘

"""

def _rdivs():
  seed(1)
  rdiv0();  rdiv1(); rdiv2(); rdiv3();
  rdiv4(); rdiv5(); rdiv6(); rdiv7()
~~~

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from options import *

#Taken verbatim from the class website.

def pairs(lst):
  last=lst[0]
  for i in lst[1:]:
    yield last,i
    last = i

def xtile(lst,lo=0,hi=0.001,width=50,
             chops=[0.1 ,0.3,0.5,0.7,0.9],
             marks=["−" ," "," ","−"," "],
             bar="|",star="*",show=" %3.0f"):
  """The function _xtile_ takes a list of (possibly)
 unsorted numbers and presents them as a horizontal
 xtile chart (in ascii format). The default is a
 contracted _quintile_ that shows the
 10,30,50,70,90 breaks in the data (but this can be
 changed− see the optional flags of the function).
 """
  def pos(p): return ordered[int(len(lst)*p)]
  def place(x):
    return int(width*float((x − lo))/(hi − lo))
  def pretty(lst):
    return ’,’.join([show % x for x in lst])
  ordered = sorted(lst)
  lo      = min(lo,ordered[0])
  hi      = max(hi,ordered[−1])
  what    = [pos(p)   for p in chops]
  where   = [place(n) for n in  what]
  out     = [" "] * width
  for one,two in pairs(where):
    for i in range(one,two):
      out[i] = marks[0]
    marks = marks[1:]
  out[int(width/2)]    = bar
  out[place(pos(0.5))] = star
  return ’’.join(out) +  "," +  pretty(what)
```

```
from sim_anneal import *
from max_walk_sat import *
```

```
#Structure from SA Lecture
import sys,re,random,math
sys.dont_write_bytecode = True

from options import *
from utils import *
from analyzer import *

myOpt = Options()

class MWS:
  debug = False

  def say(self, x):
    if self.debug:
      sys.stdout.write(str(x)); sys.stdout.flush()

  def specificRun(self, probability, klass):
    fon = klass
    XVarBest = fon.XVar
    eBest = e = 1
    eNew = 1
    k = 1
    temp = []
    self.say(int(math.fabs(eBest−1)*100))
    self.say(’ ’)

    analyze = Analyzer()
    stop = False

    for i in xrange(myOpt.mws_maxTries):
      fon.Chaos()
      for j in xrange(myOpt.mws_maxChanges):
        eNew = fon.Energy()
        if(eNew < myOpt.mws_threshold ∨ stop ≡ True):
          #% means found a solution and quit
          self.say(’%’)
          eBest = eNew
          XVarBest = list(fon.XVar)
          temp.append(eNew)
          #print xtile(temp,lo=0, hi=1, width=25,show=" %1.5f")
          return eBest, XVarBest
        else:
          #modify random part of solution
          if probability < random.uniform(0,1):
            fon.Neighbor()
            self.say(’+’)
          #maximize for some random
          else:
            fon.BestNeighbor()
          self.say(’.’)
          temp.append(eNew)
          if (i+1)*(j+1) % 40 ≡ 0 ∧ len(temp) ≠ 0:
            #print ’’
            self.say(int(math.fabs(eNew−1)*100))
            self.say(’ ’)
            #print xtile(temp,lo=0, hi=1, width=25,show=" %1.5f")
            stop = analyze.EraStop(temp)
            temp = []
      return −1, XVarBest

  def run(self, klass):
    theBest = −1
    valid = False
    eBest, XVarBest = self.specificRun(myOpt.mws_prob, klass)
    if eBest ≡ −1:
      print ’No Best Found for prob = ’, myOpt.mws_prob
      self.say(’’)
    else:
      theBest = eBest
      valid = True
    return theBest, valid

  def printOptions(self):
    print "MaxWalkSat Options:"
    print "Prob:", myOpt.mws_prob
    print "MaxTries:", myOpt.mws_maxTries, "MaxChanges", myOpt.mws_maxChanges
    print "Threshold:", myOpt.mws_threshold, "Slices:", myOpt.mws_slices
```

```
#Structure from SA Lecture
import sys,re,random,math
sys.dont_write_bytecode = True

from options import *
from utils import *
from analyzer import *

myOpt = Options()

class SA:

  def say(self, x):
    if myOpt.debug:
      sys.stdout.write(str(x)); sys.stdout.flush()

  def run(self, klass):
    sa = klass
    XVarBest = sa.XVar
    eBest = e = 1
    #print ’start energy: ’, eBest
    k = 1
    temp = []
    self.say(int(math.fabs(eBest−1)*100))
    self.say(’ ’)
    analyze = Analyzer()
    stop = False

    while k < myOpt.sa_kmax ∧ stop ≡ False:
      sa.Neighbor()
      eNew = sa.Energy()
      if eNew < eBest:
        eBest = eNew
        XVarBest = list(sa.XVar)
        self.say(’!’)

      if eNew < e:
        e = eNew
        self.say(’+’)
      #Probability Check from SA Lecture
      elif math.exp(−1*(eNew−e)/(k/myOpt.sa_kmax**myOpt.sa_cooling))<random.uniform(0,1):
      #P function should be between 0 and 1
      #more random hops early, then decreasing as time goes on
        sa.Chaos()
        self.say(’?’)
      self.say(’.’)
      k = k + 1
      temp.append(eBest)
      if k % 50 ≡ 0 ∧ k ≠ myOpt.sa_kmax ∧ len(temp) ≠ 0:
        self.say(int(math.fabs(eBest−1)*100))
        self.say(’ ’)
        #print xtile(temp,lo=0, hi=1,width=25,show=" %1.5f")
        stop = analyze.EraStop(temp)
        temp = []

    if myOpt.debug:
      #print ’\nFound best − e: ’, eBest
      #print ’Variables: ’
      for vars in XVarBest:
        self.say(vars)
        self.say(",")
      #print "\n"
    return eBest, True

  def printOptions(self):
    print "SA Options:"
    print "KMAX:", myOpt.sa_kmax, "Cooling:", myOpt.sa_cooling
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class ZDT1(Model):
  smin = 0
  smax = 1
  n = 30
  XVar = [random.uniform(smin, smax) for i in range (0, n)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    X = self.XVar
    f1 = X[0]
    g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n-1))
    f2 = g*(1-np.sqrt(X[0]/g))
    return ((f1+f2) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    X = self.XVar
    f1 = X[0]
    g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n-1))
    f2 = g*(1-np.sqrt(X[0]/g))
    return (f1+f2)

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class ZDT3(Model):
  smin = 0
  smax = 1
  n = 30
  XVar = [random.uniform(smin, smax) for i in range (0, n)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    X = self.XVar
    f1 = X[0]
    g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n−1))
    f2 = g*(1−np.sqrt(X[0]/g)−(X[0]/g)*math.sin(10*math.pi*X[0]))
    return ((f1+f2) − self.eMin) / (self.eMax − self.eMin)

  def RawEnergy(self):
    X = self.XVar
    f1 = X[0]
    g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n−1))
    f2 = g*(1−np.sqrt(X[0]/g)−(X[0]/g)*math.sin(10*math.pi*X[0]))
    return (f1+f2)

  def __init__(self):
    self.Baseline(1000)
    self.XVar = self.XVarMax
```

```
from fonseca_model import *
from schaffer_model import *
from kursawe_model import *
from ZDT1_model import *
from ZDT3_model import *
from viennet3_model import *
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class Fonseca(Model):
  n = 3
  smin = −4
  smax = 4
  XVar = [random.uniform(smin, smax) for i in range (0, 3)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    n = self.n
    f1 = 1−math.exp(−np.sum([self.XVar[i]−(1/np.sqrt(n))**2 for i in range (0, 3)]))
    f2 = 1−math.exp(−np.sum([self.XVar[i]+(1/np.sqrt(n))**2 for i in range (0, 3)]))
    return ((f1+f2) − self.eMin) / (self.eMax − self.eMin)

  def RawEnergy(self):
    n = self.n
    f1 = 1−math.exp(−np.sum([self.XVar[i]−(1/np.sqrt(n))**2 for i in range (0, 3)]))
    f2 = 1−math.exp(−np.sum([self.XVar[i]+(1/np.sqrt(n))**2 for i in range (0, 3)]))
    return f1+f2

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

myOpt = Options()

class Kursawe(Model):
  n = 3
  smin = −5
  smax = 5
  XVar = [random.uniform(smin, smax) for i in range (0, 3)]
  XVarMax = XVar
  eMax = 0
  eMin = 0
  a = 0.8
  b = 3

  def Energy(self):
    X = self.XVar
    f1 = np.sum([−10*math.exp(−0.2*(np.sqrt(X[i]**2+X[i]**2))) for i in range (0, 3−1)])
    f2 = np.sum([math.fabs(X[i])**self.a + 5*np.sin(X[i])**self.b for i in range (0, 3)])
    return ((f1+f2) − self.eMin) / (self.eMax − self.eMin)

  def RawEnergy(self):
    X = self.XVar
    f1 = np.sum([−10*math.exp(−0.2*(np.sqrt(X[i]**2+X[i]**2))) for i in range (0, 3−1)])
    f2 = np.sum([math.fabs(X[i])**self.a + 5*np.sin(X[i])**self.b for i in range (0, 3)])
    return (f1+f2)

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from options import *
myOpt = Options()
rand = random.random

class Model:
  #Default Values overwritten by subclass; should have better defaults, but...
  n = 50
  smin = 1
  smax = 1
  XVar = [random.uniform(smin, smax) for i in range (0, n)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    raise NotImplementedError()

  def RawEnergy(self):
    raise NotImplementedError()

  def Neighbor(self):
    self.XVar[random.randint(0, self.n−1)] = random.uniform(self.smin, self.smax)

  def BestNeighbor(self):
    toChange = random.randint(0, self.n−1)
    toIncrement = (self.smax − self.smin) / myOpt.mws_slices
    curMax = 1
    maxVal = self.XVar[toChange]
    for i in xrange(myOpt.mws_slices):
      self.XVar[toChange] = self.smin + toIncrement
      x = self.Energy()
      if x < curMax:
        curMax = x
        maxVal = self.XVar[toChange]

  def Reset(self):
    self.XVar = self.XVarMax

  def Chaos(self):
    for vars in self.XVar:
      vars = random.uniform(self.smin, self.smax)

  def Baseline(self, numRuns):
    self.Chaos()
    self.eMax = self.eMin = self.RawEnergy()
    runs = 1
    while runs < numRuns:
      self.Neighbor()
      eNew = self.RawEnergy()
      if eNew > self.eMax: #find largest difference
        self.eMax = eNew
        self.XVarMax = self.XVar
        #print self.XVarMax, eNew
      if eNew < self.eMin: #find smallest difference
        self.eMin = eNew
        #print ’Min: ’, self.XVar, eNew
      runs += 1
    #print ’Baseline: ’, self.eMin, ’, ’, self.eMax

  def __init__(self):
    raise NotImplementedError()
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np

from model_base import *
from options import *

sys.dont_write_bytecode = True

class Schaffer(Model):
  n = 1
  smin = -10
  smax = 10
  XVar = [random.uniform(smin, smax) for i in range (0, 1)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    f1 = self.XVar[0]*self.XVar[0]
    f2 = (self.XVar[0]-2)*(self.XVar[0]-2)
    return ((f1+f2) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    f1 = self.XVar[0]*self.XVar[0]
    f2 = (self.XVar[0]-2)*(self.XVar[0]-2)
    return (f1+f2)

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class Viennet3(Model):
  smin = -3.0
  smax = 3
  n = 2
  XVar = [random.uniform(smin, smax) for i in range (0, n)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    X = self.XVar
    f1 = 0.5*X[0]**2 + X[1]**2 + math.sin(X[0]**2+X[1]**2)
    f2 = (3*X[0]-2*X[1]+4)**2/8 + (X[0]-X[1]+1)**2/27 + 15
    f3 = 1/(X[0]+X[1]+1) - 1.1*math.e**(-X[0]**2-X[1]**2)
    return (math.fabs(f1+f2+f3) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    X = self.XVar
    f1 = 0.5*X[0]**2 + X[1]**2 + math.sin(X[0]**2+X[1]**2)
    f2 = (3*X[0]-2*X[1]+4)**2/8 + (X[0]-X[1]+1)**2/27 + 15
    f3 = 1/(X[0]+X[1]+1) - 1.1*math.e**(-X[0]**2-X[1]**2)
    return math.fabs(f1+f2+f3)

  def __init__(self):
    self.Baseline(1000)
    self.XVar = self.XVarMax
```