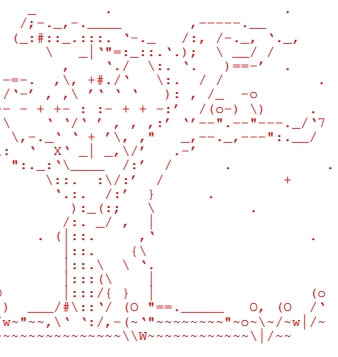
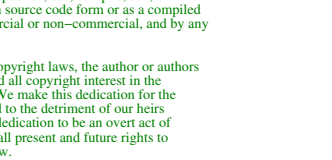
```
#!/usr/bin/env python3
# vim: ts=2 sw=2 sts=2 et :
```



```
"""
./sublime.py [OPTIONS]
(c)2022 Tim Menzies <timm@ieee.org> unlicense.org.
Sublime's unsupervised bifurcation: let's infer minimal explanations.

OPTIONS:

  -Max   max numbers to keep       : 512
  -Some  find 'far' in this many egs : 512
  -data  data file                 : ../data/auto93.csv
  -enough min leaf size            : .5
  -help  show help                 : False
  -far   how far to look in 'Some' : .9
  -p     distance coefficient      : 2
  -seed  random number seed        : 10019
  -todo  start up task             : nothing
  -xsmall Cohen's small effect     : .35

## See Also

[issues](issues) âM-^@¢ [repo](github)

## Algorithm

Stochastic clustering to generate tiny models.  Uses random projections
then  unsupervised iterative dichotomization using ranges that
most distinguish sibling clusters.

## License

This is free and unencumbered software released into the public
domain.

Anyone is free to copy, modify, publish, use, compile, sell, or
distribute this software, either in source code form or as a compiled
binary, for any purpose, commercial or non-commercial, and by any
means.

In jurisdictions that recognize copyright laws, the author or authors
of this software dedicate any and all copyright interest in the
software to the public domain. We make this dedication for the
benefit of the public at large and to the detriment of our heirs
and successors. We intend this dedication to be an overt act of
relinquishment in perpetuity of all present and future rights to
this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to <http://unlicense.org/>
"""

import random,sys,re
from random import random as r
from typing import Any
```



```
def anywhere(a:list) -> int:
  "Return a random index of list 'a'."
  return random.randint(0, len(a)-1)

big = sys.maxsize

def atom(x):
  "Return a number or trimmed string."
  x=x.strip()
  if   x=="True" : return True
  elif x=="False": return False
  else:
    try: return int(x)
    except:
      try: return float(x)
      except: return x.strip()

def demo(want,all):
  "Maybe run a demo, if we want it, resetting random seed first."
  for one in dir(all):
    if (not want or (want and one.startswith(want))):
      random.seed(the.seed)
      all.__dict__[one]()

def file(f):
  "Iterator. Returns one row at a time, as cells."
  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(",")]

def first(a:list) -> Any:
  "Return first item."
  return a[0]

def merge(old:list) -> list:
  j,n,now = -1,len(old),[]
  while j < n-1:
    j += 1
    a  = old[j]
    if j < n-2:
      if b := a.merge( old[j+1] ):
        a  = b
        j += 1
    now += [a]
  return old if len(now)==len(old) else merge(now)

class o(object):
  "Class that can pretty print its slots, with fast inits."
  def __init__(i, **d): i.__dict__.update(**d)
  def __repr__(i):
    pre = i.__class__.__name__ if isinstance(i,o) else ""
    return pre+str(
      {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})

def options(doc:str) ->o:
  """Convert 'doc' to options dictionary using command line args.
  Args canuse two 'shorthands': (1) boolean flags have no arguments (and mentioning
  those on the command line means 'flip the default value'; (2) args need only
  mention the first few of a key (e.g. -s is enough to select for -seed)."""
  d={}
  for line in doc.splitlines():
    if line and line.startswith("  -"):
       key, *_, x = line.strip()[1:].split(" ") # get 1st,last word on each line
       for j,flag in enumerate(sys.argv):
         if flag and flag[0]=="-" and key.startswith(flag[1:]):
           x= "True" if x=="False" else("False" if x=="True" else sys.argv[j+1])
       d[key] = atom(x)
  if d["help"]: exit(print(re.sub(r'\n#.*',"",doc,flags=re.S)))
  return o(**d)

def r() -> float:
  "Return random number 0..1"
  return random.random()

def second(a:list) -> Any:
  "Return second item."
  return a[1]

the = options(__doc__)
```

```python
#            ____
#           /\_  \
#           \//\  \                __        _____      _____       __       _____
#    ____     \ \  \     /'__`\   /',__\    /',__\    /'__`\    /',__\
#   /\ \__/     \_\ \_  /\ \L\.\_ /\__, `\ /\__, `\ /\  __/  /\__, `\
#   \ \____\    /\____\\ \__/.\_\\/\____/ \/\____/ \ \____\ \/\____/
#    \/____/    \/____/ \/__/\/_/ \/___/   \/___/   \/____/  \/___/

#    ___   _ __   __ _   _ _
#   (_-<  | '_ \ / _` | | ' \
#   /__/  | .__/ \__,_| |_||_|
#         |_|

class Span(o):
  """Given two ‘Sample‘s and some ‘x‘ range ‘lo..hi‘,
  a ‘Span‘ holds often that range appears in each ‘Sample‘."""
  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:
    "‘y‘ is a label identifying, one ‘Sample‘ or another."
    i.lo = min(x,i.lo)
    i.hi = max(x,i.hi)
    i.ys.add(y,inc)

  def merge(i, j)-> o|None:
    "If the merged span is simpler, return that merge."
    a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
    if 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:
    "True if the range accepts the row."
    x = row[col.at]; return x=="?" or i.lo<=x and x<i.hi

  def show(i, positive=True) -> None:
    "Show the range."
    txt = i.col.txt
    if positive:
      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}"
    else:
      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"{txt} < {i.lo} or {txt} >= {i.hi}"

  def support(i) -> float:
    "Returns 0..1."
    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
    for s in spans:
      divs.add(    s.ys.div())
      supports.add(s.support())
    return sorted(spans, key=f)

#                 _
#    ___   ___   | |
#   / _| / _ \  | |
#   \__| \___/  |_|

class Col(o):
  "Summarize columns."
  def __init__(i,at=0,txt=""):
    i.n,i.at,i.txt,i.w=0,at,txt,(-1 if "<" in txt else 1)

  def dist(i,x:Any, y:Any) -> float:
    return 1 if x=="?" and y=="?" else i.dist1(x,y)

#    _ _    _  _   _ __
#   | ' \  | || | | '  \
#   |_||_|  \_,_| |_|_|_|

class Num(Col):
  "Summarize numeric columns."
  def __init__(i,**kw):
    super().__init__(**kw)
    i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False

  def add(i,x: float ,_):
    if x != "?":
      i.n += 1
      i.lo = min(x,i.lo)
      i.hi = max(x,i.hi)
      if len(i._all) < i.max    : i.ok=False; i._all += [x]
      elif r()       < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
    return x

  def all(i):
    if not i.ok: i.ok=True; i._all.sort()
    return i._all

  def per(i,p=.5):
    a = i.all(); return a[ int(p*len(a)) ]

  def mid(i): return i.per(.5)
  def div(i): return (i.per(.9) - i.per(.1)) / 2.56

  def merge(i,j):
    k = Num(at=i.at, txt=i.txt)
    for x in i._all: k.add(x)
    for x in j._all: k.add(x)
    return k

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

  def dist1(i,x,y):
    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 spans(i,j, all):
    lo  = min(i.lo, j.lo)
    hi  = max(i.hi, j.hi)
    gap = (hi-lo) / (6/the.xsmall)
    at  = lambda z: lo + int((z-lo)/gap)*gap
    tmp = {}
    for x in map(at, i._all):
      s = tmp[x] = tmp[x] if x in tmp else Span(i,x,x+gap)
      s.add(x,0)
    for x in map(at, j._all):
      s = tmp[x] = tmp[x] if x in tmp else Span(i,x,x+gap)
      s.add(x,1)
    tmp = merge([x for _,x in sorted(tmp.items(),key=first)])
    if len(tmp) > 1 : all + tmp

#    ___   _  _   _ __
#   (_-<  | || | | '  \
#   /__/   \_, | |_|_|_|
#          |__/

class Sym(Col):
  "Summarize symbolic columns."
  def __init__(i,**kw):
    super().__init__(**kw)
    i.has, i.mode, i.most = {}, None, 0

  def add(i,x,inc):
    if x != "?":
      i.n += inc
      tmp = i.has[x] = inc + i.has.get(x,0)
      if tmp > i.most: i.most, i.mode = tmp, x
    return x

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

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

  def mid(i): return i.mode

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

  def spans(i,j, all):
    tmp = {}
    for x,n in i.has.items():
      s = tmp[x] = (tmp[x] if x in tmp else Span(i,x,x))
      s.add(x,0,n)
    for x,n in j.has.items():
      s = tmp[x] = (tmp[x] if x in tmp else Span(i,x,x))
      s.add(x,1,n)
    tmp = [second(x) for x in sorted(tmp.items(), key=first)]
    if len(tmp) > 1 : all + tmp

#                                       _
#    ___   __ _   _ __    _ __   | |  ___
#   (_-<  / _` | | '  \  | '_ \  | | / -_)
#   /__/  \__,_| |_|_|_| | .__/  |_| \___|
#                        |_|

class Sample(o):
  "Load, then manage, a set of examples."
  def __init__(i,inits=[]):
    i.rows, i.cols, i.x, i.y = [], [], [], []
    if str ==type(inits): [i + row for row in file(inits)]
    if list==type(inits): [i + row for row in inits]

  def __add__(i,a):
    def col(at,txt):
      what  = Num if txt[0].isupper() else Sym
      now   = what(at=at, txt=txt)
      where = i.y if "+" in txt or "-" in txt or "!" in txt else i.x
      if txt[-1] != ":": where += [now]
      return now
    #-----------
    if i.cols: i.rows += [[col.add(a[col.at]) for col in i.cols]]
    else:      i.cols  = [col(at,txt) for at,txt in enumerate(a)]

  def mid(i,cols=None): return [col.mid() for col in (cols or i.all)]
  def div(i,cols=None): return [col.div() for col in (cols or i.all)]

  def clone(i,inits=[]):
    out = Sample()
    out + [col.txt for col in i.cols]
    [out + x for x in inits]
    return out

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

  def far(i, x, rows=None):
    tmp= sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
    return tmp[ int(len(tmp)*the.far) ]

  def proj(i,row,x,y,c):
    a = i.dist(row,x)
    b = i.dist(row,y)
    return ((a**2 + c**2 - b**2) / (2*c) , row)

  def half(i, top=None):
    top  = top or i
    some = random.choices(i.rows, k=the.Some)
    w    = some[0]
    _,x  = top.far(w, some)
    c,y  = top.far(x, some)
    left, right = i.clone(), i.clone()
    for n,(_,r) in enumerate(
                     sorted([top.proj(r,x,y,c) for r in i.rows],key=first)):
      (left if n <= len(i.rows)//2 else right).__add__(r)
    return left,right

  def split(i,top=None):
    here = Tree(i)
    top = top or i
    if len(i.rows) >= 2*len(top.rows)**the.enough:
      left0, right0 = i.half(top)
      spans = []
      [lcol.spans(rcol,spans) for lcol,rcol in zip(left0.x, right0.x)]
      if len(spans) > 0:
        here.when   = Span.sort(spans)[0]
        left, right = i.clone(), i.clone()
        [(left if span.selects(row) else right).add(row) for row in i.rows]
        if len(left.rows) < len(i.rows): here.left  = left.split(top)
        if len(rght.rows) < len(i.rows): here.right = right.split(top)
    return here

#    _
#   | |_   _ _   ___   ___
#   |  _| | '_| / -_) / -_)
#    \__| |_|   \___| \___|

class Tree(i):
  def __init__(i,here):
    i.here, i.when, i.yes, i.no = here, None, None, None

  def show(i,pre=""):
    "Print tree with indents."
    print(f"{pre}{i.here.ys.n}")
    if i.yes:
      print(f"{pre} {i.when.show(True)}") ; i.yes.show(pre + "|.. ")
    if i.no:
      print(f"{pre} {i.when.show(False)}"); i.no.show( pre + "|.. ")
```

```
#    _
#   /\ \
#   \_\ \        __      ___ ___      ___      ____
#   /'_` \   /'__`\ /' __` __`\   / __`\  /',__\
# /\ \L\ \ /\  __/ /\ \/\ \/\ \ /\ \L\ \/\__, `\
# \ \___,_\\ \____\\ \_\ \_\ \_\\ \____/\/\____/
#   \/__,_ / \/____/ \/_/\/_/\/_/ \/___/  \/___/

class Demos:
  "Possible start-up actions."
  def opt():
    print(the)

  def num():
    n=Num()
    for x in range(10000): n.add(x)
    print(sorted(n._all),n)

  def sym():
    s=Sym()
    for x in range(10000): s.add( int(r()*20))
    print(s)

  def rows():
    for row in file(the.data): print(row)

  def sample(): s=Sample(the.data); print(len(s.rows))

  def done(): s=Sample(the.data); s.dist(s.rows[1], s.rows[2])

  def dist():
    s=Sample(the.data)
    for row in s.rows: print(s.dist(s.rows[0], row))

  def far():
    s=Sample(the.data)
    for row in s.rows: print(row,s.far(row))

  def clone():
    s=Sample(the.data); s1 = s.clone(s.rows)
    print(s.x[0])
    print(s1.x[0])

  def half():
    s=Sample(the.data); s1,s2 = s.half()
    print(s1.mid(s1.y))
    print(s2.mid(s2.y))

if __name__ == "__main__":
  demo(the.todo,Demos)
```