~~~python
#!/usr/bin/env python3
# vim: ts=2 sw=2 sts=2 et :
#
#     dew   ~~~~~~~~~~~~~~~~~~~~~~~\\W~~~~~~~~~~~~\|/~~
#               ~~~w/w~"~~,\` `:/,-(~`"~~~~~~~~~"~o~\~/~w|/~
#                    )  ___/#\::`/ (O "==._____   O,  (O /`
#                   O       |:::/{ }  |                  (o
#                           |:::(\    |
#                           |::.\  \ `.
#          .                |::.    {\
#                        . (|::.     ,`                  .
#                           /:. _/ ,  |
#                .           ):_(:;   \           .
#        .                 `.:.  /:'  }      .
#                         \::.  :\/:'  /              +
#     .               ":._:`\_____ /:'  /      .           .
#                   \:  `  X` _| _,\/'   .-'
#        `.   (    \: \,-._` ` + '\, ,"   _,--._,---":.___/
#       .      ,=':  \    ` `/` ' , , ,:' `'--".--"---._/`7
#            .    /:+- - + +- : :- + + -:'  /(o-) \)     .
#     .           \/:/`-' , ,\ '` ` `   ):  , /_   -o
#         .      ., ,-=-.  ,\, +#./`   \:.  / /           .
#                           ,    `./  \:. `.   )==-'  .
#       `                 \   _|`"=:_::.`.);  \ ___/ /
#       ((       .    (_:#::_.:::. `-._   /:, /-._, `._,
#       ,              /;-._,-.____        ,-----.__
#          .           _        .                     .
#                 .        +          .      .          .
"""
./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
    -cautious  On any crash, stop+show stack : False
    -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](https://github.com/timm/sublime/issues)
:: [repo](https://github.com/timm/sublime)
:: [repo](sublime)
::  [srx](sublime)


## Algorithm

Stochastic clustering to generate tiny models.  Uses random projections
to divide the space. Then, optionally, explain the clusters by
unsupervised iterative dichotomization using ranges that most
distinguish sibling clusters.

e.g.1: just bi-cluster on two distant points

```
/sublime.py -c -s $RANDOM -t cluster
                                        :   398
|..                                     :   199
|.. |..                                 :    99
|.. |.. |..                             :    49   Lbs-   Acc+  Mpg+
|.. |.. |.. |..                         :    24  : [2255, 15.5, 30]
|.. |.. |.. |..                         :    25  : [2575, 16.4, 30]
|.. |.. |..                             :    50
|.. |.. |.. |..                         :    25  : [2110, 16.4, 30] <== best
|.. |.. |.. |..                         :    25  : [2205, 16, 30]
|.. |..                                 :   100
|.. |.. |..                             :    50
|.. |.. |.. |..                         :    25  : [2234, 15.5, 30]
|.. |.. |.. |..                         :    25  : [2278, 16.5, 30]
|.. |.. |..                             :    50
|.. |.. |.. |..                         :    25  : [2220, 15.5, 30]
|.. |.. |.. |..                         :    25  : [2320, 15.8, 30]
|..                                     :   199
|.. |..                                 :    99
|.. |.. |..                             :    49
|.. |.. |.. |..                         :    24  : [2451, 16.5, 20]
|.. |.. |.. |..                         :    25  : [3021, 15.5, 20]
|.. |.. |..                             :    50
|.. |.. |.. |..                         :    25  : [3425, 17.6, 20]
|.. |.. |.. |..                         :    25  : [3155, 16.7, 20]
|.. |..                                 :   100
|.. |.. |..                             :    50
|.. |.. |.. |..                         :    25  : [4141, 13.5, 10]
|.. |.. |.. |..                         :    25  : [4054, 13.2, 20]
|.. |.. |..                             :    50
|.. |.. |.. |..                         :    25  : [4425, 11, 10]
|.. |.. |.. |..                         :    25  : [4129, 13, 10]
```

e.g. #2, as above but split on range that mist divides data

```
./sublime.py -c -s $RANDOM -t xplain
                                                    Lbs-  Acc+  Mgg+
                                        :   398 : [2807, 15.5, 20]
198 <= Lbs < 454                        :   167 : [3725, 14.5, 20]
|.. Modl < 72                           :    34 : [3609, 13,   20]
|.. Modl >= 72                          :   133 : [3735, 14.9, 20]
|.. |.. Cylr < 8                        :    56 : [3336, 17,   20]
|.. |.. |.. 77 <= Modl < 82             :    22 : [3410, 17.1, 20]
|.. |.. |.. Modl < 77 or Modl >= 82     :    34 : [3233, 17,   20]
|.. |.. Cylr >= 8                       :    77 : [4129, 13.2, 20]
|.. |.. |.. Modl < 75                   :    37 : [4274, 13,   10]
|.. |.. |.. Modl >= 75                  :    40 : [3962, 13.5, 20]
|.. |.. |.. |.. Lbs >= 302              :    35 : [4054, 13.2, 20]
Lbs < 198 or Lbs >= 454                 :   231 : [2290, 16,   30] <== best
```

## 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 traceback, random, math, sys, re
from random import random as r
from typing import Any
~~~

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

def any(a:list) -> Any:
  "Return a random item."
  return a[anywhere(a)]

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(do,all):
  "Maybe run a demo, if we want it, resetting random seed first."
  todo = dir(all)
  if do and do != "all":
    todo = [x for x in dir(all) if x.startswith(do)]
  for one in todo:
    fun = all.__dict__.get(one,"")
    if type(fun)==type(demo):
      random.seed(the.seed)
      doc = re.sub(r"\n\s+", "\n", fun.__doc__ or "")
      try:
        fun()
        print("PASS:", doc)
      except Exception as e:
        all.fails += 0
        if the.cautious : traceback.print_exc(); exit(1)
        else            : print("FAIL:", doc, e)
  exit(all.fails)

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(b4:list) -> list:
  "While we can find similar adjacent things, merge them."
  j,n,now = -1,len(b4),[]
  while j < n-1:
    j += 1
    a  = b4[j]
    if j < n-2:
      if merged := a.merge(b4[j+1]):
        a  = merged
        j += 1 # we will continue, after missing one
    now += [a]
  # if ‘now‘ is same size as ‘b4‘, look for any other merges.
  return b4 if len(now)==len(b4) 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 rn(x:float, n=3) -> float:
  "Round a number to three decimals."
  return round(x,n)

def rN(a:list, n=3) -> list:
  "Round a list of numbers to three decimals."
  return [rn(x,n=n) for x in a]

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

```
#           ___
#          /\_ \
#   ____   \//\ \        __       ____    ____     __     ____
#  /'___\    \ \ \     /'__`\    /',__\  /',__\  /'__`\  /',__\
# /\ \___/    \_\ \_  /\ \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): # -> Span|None
    "If the merged span is simpler, return that merge."
    a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
    if (i.ys.n==0 or j.ys.n==0 or
        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[i.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)**.5/2**.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 Sym(Col):
  "Summarize symbolic columns."
  def __init__(i,**kw):
    super().__init__(**kw)
    i.has, i.mode, i.most = {}, None, 0

  def add(i, x:str, inc:int=1) -> str:
    "Update symbol counts in ‘has‘, updating ‘mode‘ as we go."
    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:str, y:str) ->float:
    "Distance between two symbols."
    return 0 if x==y else 1

  def div(i):
    "Return diversity of this distribution (using entropy)."
    p = lambda x: x / (1E-31 + i.n)
    return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )

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

  def mid(i):
    "Return central tendancy of this distribution (using mode)."
    return i.mode

  def spans(i,j, out):
    """For each symbol in ‘i‘ and ‘j‘, count the
 number of times we see it on either side."""
    xys = [(x,"this",n) for x,n in i.has.items()] + [
           (x,"that",n) for x,n in j.has.items()]
    one, last = None,None
    all  = []
    for x,y,n in sorted(xys, key=first):
      if x != last:
         last = x
         one  = Span(i, x,x)
         all += [one]
      one.add(x,y,n)
    if len(all) > 1 : out += all
#  _ _   _  _   _ __
# | ' \ | || | | '  \
# |_||_| \_,_| |_|_|_|

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 ,inc=1):
    "Reservoir sampler. If ‘_all‘ is full, sometimes replace an item at random."
    if x != "?":
      i.n += inc
      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):
    "Return ‘_all‘, sorted."
    if not i.ok: i.ok=True; i._all.sort()
    return i._all

  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 div(i):
    """Report the diversity of this distribution (using standard deviation).
  &pm;2, 2,56, 3 &sigma; is 66,90,95%, of the mass.  28&sigma;. So one
  standard deviation is (90–10)th divide by  2.4 times &sigma;."""
    return (i.per(.9) - i.per(.1)) / 2.56

  def merge(i,j):
    "Return two ‘Num‘s."
    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 mid(i):
    "Return central tendency of this distribution (using median)."
    return i.per(.5)

  def norm(i,x):
    "Normalize ‘x‘ to the range 0..1."
    return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)

  def per(i,p:float=.5) -> float:
    "Return the p-th ranked item."
    a = i.all(); return a[ int(p*len(a)) ]

  def spans(i,j, out):
    """Divide the whole space ‘lo‘ to ‘hi‘ into, say, ‘xsmall‘=16 bin,
  then count the number of times we the bin on other side.
  Then merge similar adjacent bins."""
    lo  = min(i.lo, j.lo)
    hi  = max(i.hi, j.hi)
    gap = (hi-lo) / (6/the.xsmall)
    xys = [(x,"this",1) for x in i._all] + [
           (x,"that",1) for x in j._all]
    one = Span(i,lo,lo)
    all = [one]
    for x,y,n in sorted(xys, key=first):
      if one.hi - one.lo > gap:
        one  = Span(i, one.hi,x)
        all += [one]
      one.add(x,y,n)
    all = merge(all)
    all[ 0].lo = -big
    all[-1].hi =  big
    if len(all) > 1: out += all
#                     _          _
#  ___  __ __  _ __  | |  __ _  (_)  _ _
# / -_) \ \ / | '_ \ | | / _` | | | | ' \
# \___| /_\_\ | .__/ |_| \__,_| |_| |_||_|
#             |_|

class Explain(o):
  "Tree with ‘yes‘,‘no‘ branches for samples that do/do not match a ‘span‘."
  def __init__(i,here):
    i.here, i.span, i.yes, i.no = here, None, None, None

  def show(i,pre=""):
    if not pre:
      tmp= i.here.mid(i.here.y)
      print(f"{’’:40} : {len(i.here.rows):5} : {tmp}")
    if i.yes:
      s=f"{pre}{i.span.show(True)}"
      tmp= i.yes.here.mid(i.yes.here.y)
      print(f"{s:40} : {len(i.yes.here.rows):5} : {tmp}")
      i.yes.show(pre + "|.. ")
    if i.no:
      s=f"{pre}{i.span.show(False)}"
      tmp= i.no.here.mid(i.no.here.y)
      print(f"{s:40} : {len(i.no.here.rows):5} : {tmp}")
      i.no.show(pre + "|.. ")

#        _                _
#   __  | |  _  _   ___  | |_   ___   _ _
#  / _| | | | || | (_-<  |  _| / -_) | '_|
#  \__| |_|  \_,_| /__/   \__| \___| |_|
#
class Cluster(o):
  "Tree with ‘left‘,‘right‘ samples, broken at median between far points."
  def __init__(i,here,x=None,y=None,c=None,mid=None):
    i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None

  def show(i,pre=""):
    s= f"{pre:40} : {len(i.here.rows):5}"
    print(f"{s}" if i.left  else f"{s} : {i.here.mid(i.here.y)}")
    for kid in [i.left,i.right]:
      if kid: kid.show(pre + "|.. ")
#                              _
#  ___   __ _   _ __    _ __  | |  ___
# (_-<  / _` | | '  \  | '_ \ | | / -_)
# /__/  \__,_| |_|_|_| | .__/ |_| \___|
#                      |_|

class Sample(o):
  "Load, then manage, a set of examples."
  def __init__(i,inits=[]):
    i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [],None
    if str ==type(inits): [i.add(row) for row in file(inits)]
    if list==type(inits): [i.add(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]
        if "!" in txt: i.klass = 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 clone(i,inits=[]):
    out = Sample()
    out.add([col.txt for col in i.cols])
    [out.add(x) for x in inits]
    return out

  def cluster(i,top=None):
    """Split the data using random projections. Find the span that most
  separates the data. Divide data on that span."""
    here = Cluster(i)
    top = top or i
    if len(i.rows) >= 2*(len(top.rows)**the.enough):
      left,right,x,y,c,mid = i.half(top)
      if len(left.rows) < len(i.rows):
        here       = Cluster(i,x,y,c,mid)
        here.left  = left.cluster(top)
        here.right = right.cluster(top)
    return here

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

  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 half(i, top=None):
    "Using two faraway points ‘x,y‘ break data at median distance."
    some= i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
    top= top or i
    w  = any(some)
    _,x= top.far(w, some)
    c,y= top.far(x, some)
    tmp= [r for _,r in sorted([(top.proj(r,x,y,c),r)
                               for r in i.rows],key=first)]
    mid= len(tmp)//2
    return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]

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

  def proj(i,row,x,y,c):
    "Find the distance of a ‘row‘ on a line between ‘x‘ and ‘y‘."
    a = i.dist(row,x)
    b = i.dist(row,y)
    return (a**2 + c**2 - b**2) / (2*c)

  def xplain(i,top=None):
    """Split the data using random projections. Find the span that most
  separates the data. Divide data on that span."""
    here = Explain(i)
    top = top or i
    tiny = len(top.rows)**the.enough
    if len(i.rows) >= 2*tiny:
      left, right,*_ = i.half(top)
      spans = []
      [lcol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
      if len(spans) > 0:
        here.span = Span.sort(spans)[0]
        yes, no = i.clone(), i.clone()
        [(yes if here.span.selects(row) else no).add(row) for row in i.rows]
        if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
        if tiny <= len(no.rows ) < len(i.rows): here.no  = no.xplain(top=top)
    return here
```

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

class Demos:
  "Possible start-up actions."
  fails=0
  def opt():
    "show the config."
    [print(f"{k:>10} = {v}") for k,v in the.__dict__.items()]

  def seed():
    "seed"
    assert .494 <= r() <= .495

  def num():
    "check `Num`."
    n = Num()
    for _ in range(100): n.add(r())
    assert .30 <= n.div() <= .31, "in range"

  def sym():
    "check `Sym`."
    s = Sym()
    for x in "aaaabbc": s.add(x)
    assert 1.37 <= s.div() <= 1.38, "entropy"
    assert 'a'  == s.mid(), "mode"

  def rows():
    "count rows in a file."
    assert 399 == len([row for row in file(the.data)])

  def sample():
    "sampling."
    s = Sample(the.data)
    assert 398 == len(s.rows),     "length of rows"
    assert 249 == s.x[-1].has[1], "symbol counts"

  def dist():
    "distance between rows"
    s = Sample(the.data)
    assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842

  def far():
    "distant items"
    s = Sample(the.data)
    for _ in range(32):
      a,_ = s.far(any(s.rows))
      assert a>.5, "large?"

  def clone():
    "cloning"
    s = Sample(the.data)
    s1 = s.clone(s.rows)
    d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
    for k,v in d1.items():
      assert d2[k] == v, "clone test"

  def half():
    "divide data in two"
    s = Sample(the.data); s1,s2,*_ = s.half()
    print(s1.mid(s1.y))
    print(s2.mid(s2.y))

  def cluster():
    "divide data in two"
    s = Sample(the.data)
    s.cluster().show(); print("")

  def xplain():
    "divide data in two"
    s = Sample(the.data)
    s.xplain().show(); print("")

#---------------------------------------------------
the=options(__doc__)
if __name__ == "__main__": demo(the.todo,Demos)

"""
all config local to Sample
Example class
"""
```