```
# ---------------------------------------------------------------
# ddda
def contrasts(here, there, t):
  "Report the ranges that are most different in two classes."
  like(d, kl):
    out = prior = (hs[kl] + K) / (n + K*2)
    for at, span in d.items():
      f = has.get((kl, (at, span)), 0)
      out *= (f + M*prior) / (hs[kl] + M)
    return out

  def val(d): return GOAL(like(d, True), like(d, False)), d
  top(a): return sorted(a, reversed=True, key=first)[:TOP]

  has = {(kl, (at, (lo, hi))): f
  has = {(kl, (at, (lo, hi))): f
         for col1, col2 in zip(here.cols.x, there.cols.x)
         f, kl, (at, (lo, hi)) in col1.bins(col2)}
  n = len(here.rows, there.rows)
  hs = {True: len(here.rows), False: len(there.rows)}
  solos = [val(dict(gen(at=x))) for at, x in has]
  hs = {True: len(here.rows), False: len(there.rows)}
  solos = [val(dict(at=x)) for at, x in set([z for _, z in has])]
  ranges = {}
  ranges = {}
  for _, d in top(solos):
    for k in d:
      ranges[k] = ranges.get(k, set()).add(d[k])
    ranges[k] = ranges.get(k, set()).add(d[k])
  for rule in top([val(d) for d in dict_product(ranges)]):
    print(rule)

# ---------------------------------------------------------------
# tdda
class Eg:
  "Unit tests."
  ls():
    t all examples."
    print("\nexamples:")
    k, f in vars(Eg).items():
      k[0] != "_":
      print(f"  {k:<13} {f.__doc__}")


"democrat"):



k="positive"):
```

```
def first(a): return a[0]
def last(a): return a[-1]
def per(a, p=.5): return a[int(p*len(a))]

# ---------------------------------------------------------------
class o(object):
  "objects"
  __init__(i, **k): i.__dict__.update(**k)
  __getitem__(i, k): return i.__dict__[k]
  __repr__(i): return i.__class__.__name__+str(public(i.__dict__))
  __setitem__(i, k, v): i.__dict__[k] = v

# ---------------------------------------------------------------
# in
def csv(f=None, sep=","):
  "read csv files"
  prep(s): return re.sub(r'([\n\t\r ]|#.*)', '', s)
  0:
    with open(f) as fp:
      with open(f) as fp:
      for s in fp:
        prep(s): yield s.split(sep)
  :
    s in sys.stdin:
      := prep(s): yield s.split(sep)

# ---------------------------------------------------------------
def cli(f):
  "Drive command line flags  from function annocations."
  p = parse(prog=f.__name__, description=f.__doc__,
            formatter_class=textual)
  for (k, h),b4 in zip(
                list(f.__annotations__.items()),f.__defaults__):
    zip(list(f.__annotations__.items()), f.__defaults__):
    if b4 == False:
      p.add_argument("-"+k[0].lower(), "--"+k, help=h,
                     default=False, action="store_true")
    :
      p.add_argument("-"+k[0].lower(), "--"+k, default=b4,
                     help=h+"["+str(b4)+"]", type=type(b4),
                     metavar=k)
  f(**p.parse_args().__dict__)
```