```
USAGE: lua eg.lua [OPTIONS]

OPTIONS:
 -b  --bins    number of bins            = 8
 -c  --cohen   small effect              = .35
 -e  --eg      start-up example          = nothing
 -F  --far     far away                  = .95
 -f  --file    file with csv data        = ../../docs/auto93.csv
 -h  --help    show help                 = false
 -m  --min     min size = n^(the.min)    = .5
 -n  --nums    how many numbers to keep  = 256
 -p  --p       distance coeffecient      = 2
 -s  --seed    random number seed        = 10019
 -S  --sample  how many rows to search   = 512]])
```

Commonly used lib functions.

```
local lt,o,oo,map   = l.lt,l.o,l.oo,l.map
local per,push,sort = l.per, l.push,l.sort

---- ---- ---- ---- Classes
local Data,Cols,Sym,Num,Row
```

Holder of rows and their sumamries (in cols).

```
function Data() return {_is = "Data",
                        cols= nil,  -- summaries of data
                        rows= {}    -- kept data
                       } end
```

Holds of summaries of columns. Columns are created once, then may appear in multiple slots.

```
function Cols() return {
  _is  = "Cols",
  names={},  -- all column names
  all={},    -- all the columns (including the skipped ones)
  klass=nil, -- the single dependent klass column (if it exists)
  x={},      -- independent columns (that are not skipped)
  y={}       -- depedent columns (that are not skipped)
  } end
```

Summarizers a stream of symbols.

```
function Sym(c,s)
  return {_is= "Sym",
          n=0,           -- items seen
          at=c or 0,     -- column position
          name=s or "", -- column name
          _has={}        -- kept data
         } end
```

Summarizes a stream of numbers.

```
function Num(c,s)
  return {_is="Nums",
          n=0,at=c or 0, name=s or "", _has={}, -- as per Sym
          isNum=true,      -- mark that this is a number
          lo= math.huge,   -- lowest seen
          hi= -math.huge,  -- highest seen
          isSorted=true,    -- no updates since last sort of data
          w=(s or ""):find"-$" and -1 or 1 -- minimizing if w=-1
         } end
```

Holds one record

```
function Row(t) return {_is="Row",
                        cells=t,          -- one record
                        cooked=l.copy(t), -- used if we discretize data
                        isEvaled=false    -- true if y-values evaluated.
                       } end

---- ---- ---- ---- Data Functions
local add,adds,clone,div,mid,norm,nums,record,records,stats
```

—- —- —-. Create Generate rows from some src. If src is a string, read rows from file; else read rows from a src table. When reading, use row1 to define columns.

```
function records(src,      data,head,body)
  function head(sNames)
    local cols = Cols()
    cols.names = namess
    for c,s in pairs(sNames) do
      local col = push(cols.all, -- Numerics start with Uppercase.
                       (s:find"^[A-Z]*" and Num or Sym)(c,s))
      if not s:find":$" then -- some columns are skipped
        push(s:find"[!+-]" and cols.y or cols.x, col) -- some cols are goal cols
        if s:find"!$"    then cols.klass=col end end end
    return cols
  end ------------
  function body(t) -- treat first row differently (defines the columns)
    if data.cols then record(data,t) else data.cols=head(t) end
  end ----------
  data =  Data()
  if type(src)=="string" then l.csv(src, body) else
    for _,t in pairs(src or {}) do body(t) end end
  return data end
```

Return a new data with same structure as data1. Optionally, oad in rows.

```
function clone(data1,  rows)
  data2=Data()
  data2.cols = _head(data1.cols.names)
  for _,row in pairs(rows or {}) do record(data2,row) end
  return data2 end

---- ---- ---- Update
```

Add one thing to col. For Num, keep at most nums items.

```
function add(col,v)
  if v~="?" then
    col.n = col.n + 1
    if not col.isNum then col._has[v] = 1 + (col._has[v] or 0) else
       col.lo = math.min(v, col.lo)
       col.hi = math.max(v, col.hi)
       local pos
       if     #col._has < the.nums            then pos = 1 + (#col._has)
       elseif math.random() < the.nums/col.n then pos = math.random(#col._has) end
       if pos then col.isSorted = false
                   col._has[pos] = tonumber(v) end end end end
```

Add many things to col

```
function adds(col,t) for _,v in pairs(t) do add(col,v) end; return col end
```

Add a row to data. Calls add() to updatie the cols with new values.

```
function record(data,xs)
  local row= push(data.rows, xs.cells and xs or Row(xs)) -- ensure xs is a Row
  for _,todo in pairs{data.cols.x, data.cols.y} do
    for _,col in pairs(todo) do
      add(col, row.cells[col.at]) end end end

---- ---- ---- Query
```

Return kept numbers, sorted.

```
function nums(num)
  if not num.isSorted then num._has = sort(num._has); num.isSorted=true end
  return num._has end
```

Normalized numbers 0..1. Everything else normalizes to itself.

```
function norm(col,n)
  return x=="?" or not col.isNum and x or  (n-col.lo)/(col.hi-col.lo + 1E-32) end
```

Diversity (standard deviation for Nums, entropy for Syms)

```
function div(col)
  if  col.isNum then local a=nums(col); return (per(a,.9)-per(a,.1))/2.58 else
    local function fun(p) return p*math.log(p,2) end
    local e=0
    for _,n in pairs(col._has) do if n>0 then e=e-fun(n/col.n) end end
    return e end end
```

Central tendancy (median for Nums, mode for Syms)

```
function mid(col)
  if col.isNum then return per(nums(col),.5) else
    local most,mode = -1
    for k,v in pairs(col._has) do if v>most then mode,most=k,v end end
    return mode end end
```

For showCols (default=data.cols.x) in data, report fun (default=mid).

```
function stats(data,  showCols,fun,    t)
  showCols, fun = showCols or data.cols.y, fun or mid
  t={}; for _,col in pairs(showCols) do t[col.name]=fun(col) end; return t end

---- ---- ---- ---- Discretization
local bins,cook,divs
```

Find ranges within a num (unsupervised).

```
function bins(num)
  local a, epsilon = nums(num), the.cohen*div(num)
  local enough = #a^the.min
  local one = {lo=a[1], hi=a[1], n=0}
  local t = {one}
  for i,x in pairs(a) do
    if i < #a-enough and x ~= a[i+1] and n > enough and hi-lo > epsilon then
      one = push(t, {lo=one.hi, hi=a[i], n=0})  end
    one.hi = a[i]
    one.n  = 1 + one.n end
  t[1].lo  = -math.huge
  t[#t].ho =  math.huge
  return t end
```

Fill in discretized values (in cooked).

```
function cook(data)
  for _,num in pairs(data.cols.x) do
    if num.isNum then local t = bins(num)
                      for _,row in pairs(data.rows) do
                        local v = row.cells[num.at]
                        if v ~= "?" then
                          for _,bin in pairs(t) do
                            if v > bin.lo and v <= bin.hi then
                              row.cooked[col.at] = bin.lo
                              break end end end end end end
```

Sum the entropy of the coooked independent columns.

```
function divs(data,rows)
  local n = 0
  for _,col in pairs(data.cols.x) do
    local sym= Sym()
    for _,row in pairs(rows or data.rows) do
      v = row.cooked[col.at]
      if v ~= "?" then add(s, v) end end
    n = n + div(sym) end
  return n end

---- ---- ---- Distance functions
local around, dist, far, half, halves, tree
```

Distance between rows (returns 0..1). For unknown values, assume max distance.

```
function dist(data,t1,t2)
  local function fun(col,  v1,v2)
    if   v1=="?" and v2=="?" then return 1 end
    if not col.isNum then return v1==v2 and 0 or 1 end
    v1,v2 = norm(col,v1), norm(col,v2)
    if v1=="?" then v1 = v2<.5 and 1 or 0 end
    if v2=="?" then v2 = v1<.5 and 1 or 0 end
    return math.abs(v1-v2)
  end -------
  local d = 0
  for _,col in pairs(data.cols.x) do
    d = d + fun(col, t1.cells[col.at], t2.cells[col.at])^the.p end
  return (d/#data.cols.x)^(1/the.p) end
```

Sort rows (default=data.rows) by distance to row1.

```
function around(data,row1,  rows,     fun)
  function fun(row2) --print("r2",#row2);
     return {row=row2, dist=dist(data,row1,row2)} end
  print("a")
  return sort(map(rows or data.rows,fun),lt"dist") end
```

Return the row that is the.far to max distance away from row.

```
function far(data,row,  rows)
  print("f",data._is,#rows,o(row))
  return per(around(data,row,rows), the.far).row end
```

Split rows (default=data.rows) in half by distance to 2 distant points.

```
function half(data,rows,  rowAbove)
  local rows  = rows or data.rows
  local some  = l.many(rows, the.sample)
  local left  = rowAbove or far(data, l.any(some),some)
  print(4,data._is,o(left),#some)
  local right = far(data, left,some)
  print(5)
  local c     = dist(data,left,right)
  local lefts,rights = {},{}
  local function fun(row)
                  local a = dist(data,row,left)
                  local b = dist(data,row,right)
                  return {row=rows, d=(a^2 + c^2 - b^2) / (2*c)} end
  for i,rowd in pairs(sort(map(rows, fun), lt"d")) do
    push(i <= (#rows)/2 and lefts or rights, rowd.row) end
  return left,right,lefts,rights,c end

function halves(data,rows,  stop,rowAbove)
  rows = rows or data.rows
  stop = stop or (#rows)^the.min
  if #rows <= stop then return {node=rows} end
  local left,right,lefts,rights,_ = half(data,rows,rowAbove)
  return {node=rows, kids={halves(data,lefts,stop,left),
                           halves(data,rights,stop,right)}} end

function tree(x,  nodeFun,     pre)
  nodeFun = nodeFun or io.write
  pre = pre or "|.. "
  print(pre,nodeFun(x.node))
  for _,kid in pairs(x.kids or {}) do tree(kid, nodeFun, pre.."|.. ") end end
```

That's all folks.

```lua return { the=the, Data=Data, Cols=Cols, Sym=Sym, Num=Num, Row=Row, add=add, adds=adds, around=around,
bin=bins,clone=clone,cook=cook,dist=dist, div=div, divs=divs, far=far, half=half, halves=halves, mid=mid, nums=nums,
records=records, record=record, stats=stats}
```