```lisp
; vim: ts=2 sw=2 et :
(defstruct  about
  (what       "wicked.lisp do cool words")
  (copyright "(c) 2022 Tim Menzies")
  (file       "where to get data          ")
  (cohen      "small effect size          ")
  (far        "how far to seek distant items ")
  (help       "show help                  ")
  (some       "max samples for distance calcs")
  (lnorm      "norm; –L 2 means 'euclidean'  ")
  (seed       "random number seed         ")
  (cli        '(file cohen far help some lnorm seed)))

(defvar *about* (make-about
  :cohen .35 :far .9 :help  nil :some 256 :lnorm 2 :seed 10019
  :file "../data/auto93.csv" ))

(defmethod print-object ((x about) s)
  (let ((ab (make-about)))
    (format s "~&~a~%~a~%~%OPTIONS:~%" (about-what ab) (about-copyright ab))
    (dolist (y (about-cli ab))
      (format s " ––~(~8a~) ~4a ~a~%" y (slot-value ab y) (slot-value x y)))))

;;;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
;  ____ _  _ _  _ ____ ___ _ ____ _  _ ____
;  |___ |  | |\ | |     |  | |  | |\ | [__
;  |    |__| | \| |___  |  | |__| | \| ___]

(defmacro ? (x) `(slot-value *about* ',x))

(defmacro o (s x &rest xs)
  (if xs `(o (slot-value ,s ',x) ,@xs) `(slot-value ,s ',x)))

(defun sum (lst &optional (f #'identity)) (reduce '+ (mapcar f lst)))

(defmacro has (key dictionary)
  `(cdr (or (assoc ,key ,dictionary :test 'equal)
            (car (setf ,dictionary (cons (cons ,key 0) ,dictionary))))))

(defun words (s &optional (sep #\,) (x 0) (y (position sep s :start (1+ x))))
  (cons (subseq s x y) (and y (words s sep (1+ y)))))

(defun thing (s)
  (let ((s (string-trim '(#\Space #\Tab) s)))
    (if (equal s "?") #\?
      (let ((x (ignore-errors (read-from-string s)))) (if (numberp x) x s)))))

(defmacro with-csv ((cells file &optional out) &body body)
  (let ((s (gensym)))
    `(let (,cells)
       (with-open-file (,s ,file)
         (loop while (setf ,cells (read-line ,s nil)) do
           (setf ,cells (mapcar 'thing (words ,cells)))
           ,@body) ,out))))

(defmacro do-cells ((at cell rows &optional out) &body body)
  (let ((row (gensym)))
    `(let (,cell) (dolist (,row ,rows ,out)
                    (setf ,cell (elt (slot-value ,row 'cells) ,at))
                    (when (not (equal ,cell #\?)) ,@body)))))

(defun cli ()
  (labels ((args ()     #+clisp ext:*args* #+sbcl sb-ext:*posix-argv*)
           (has  (x)    (member x (args) :test 'equalp))
           (new  (x b4) (if (has x) (cond ((eq b4 t)   nil)
                                          ((eq b4 nil) t)
                                          (t (thing (elt (has x) 1))))
                          b4)))
    (dolist (slot (? cli) *about*)
      (setf (slot-value *about* slot)
            (new (format nil "––~(~a~)" slot) (slot-value *about* slot))))))

;;;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
;  ____ ___ ____ _  _ ____ ___ ____
;  [__   |  |__/ |  | |     |  [__
;  ___]  |  |  \ |__| |___  |  ___]

;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
(defstruct row (at 0) (txt "") egs cells about)

(defmethod at ((self row) n) (elt (at self cells) n))

(defmethod dist ((x row) (y row))
  (labels ((inc (col) (dist col (at x (o col at)) (at y (o col at)))))
    (let* ((xs (o self egs cols x))
           (n  (length xs))
           (p  (? lnorm))
           (d  (sum xs (lambda (x) (expt (inc x) p)))))
      (expt (/ d n) (/ 1 p)))))

;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
(defstruct (num (:constructor %make-num))
  (n 0) (at 0) (w 0) (txt "") (mid 0) (div 0) (m2 0) (lo 1E32) (hi -1E32))

(defun make-num (at txt rows &aux (it (%make-num :at at :txt txt)))
  (with-slots (w n most mid div m2) it
    (setf w (if (lessp txt) -1 1))
    (do-cells (at x rows it)
      (let ((d (- x mid)))
        (incf n)
        (incf mid (/ d n))
        (incf m2  (* d (- x mid)))
        (setf div  (if (< n 2) 0 (sqrt (/ m2 ( - n 1)))))))))

(defmethod dist ((self num) x y)
  (cond ((and (eq x #\?) (eq y #\?)) (return-from dist 1))
        ((eq x #\?) (setf y (norm self y) x (if (< y .5) 1 0)))
        ((eq y #\?) (setf x (norm self x) y (if (< x .5) 1 0)))
        (t          (setf x (norm self x) y (norm self y))))
  (abs (- x y)))
;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
(defstruct (sym (:constructor %make-sym))
  (n 0) (at 0) (txt "") all (most 0) mid (div 0))

(defun make-sym (at txt rows &aux (it (%make-sym :at at :txt txt)))
  (with-slots (n all most mid div) it
    (do-cells (at x rows it)
      (incf n)
      (let ((tmp (incf (has x all))))
        (if (> tmp most)
          (setf most tmp
                mid  x))))
    (dolist (two all)
      (print two)
      (let ((p (/ (cdr two) n))) (decf div (* p (log p 2)))))))

(defmethod dist ((self sym) x y)
  (if (and (eq x #\?) (eq y #\?)) 0 (if (equal x y) 0 1)))


;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
(defstruct (cols (:constructor %make-cols)) klass all x y names)

(labels ((pre=   (s x) (eql x (char s 0)))
         (end=   (s x) (eql x (char s (1- (length s))))))
  (defun ignorep (s)   (end= s #\X))
  (defun klassp  (s)   (end= s #\!))
  (defun lessp   (s)   (end= s #\-))
  (defun morep   (s)   (end= s #\+))
  (defun goalp   (s)   (or (end= s #\+) (end= s #\-) (end= s #\!)))
  (defun nump    (s)   (if (pre= s #\$) 'make-num 'make-sym)))

(defun make-cols (egs names &aux (it (%make-cols :names names)))
  (with-slots (all x y names klass) it
    (loop for txt in names for at from 0 do
      (let ((col (funcall (if (nump) 'make-num 'make-sym) at txt egs)))
        (push col all)
        (when (not (ignorep txt))
          (if (klassp txt) (setf klass col))
          (if (goalp txt)  (push col y) (push col x))))))
   it)
;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
(defstruct (egs (:constructor %make-egs)) rows cols)

(defun make-egs (rows &aux (it (%make-egs)))
 (setf (egs-cols it) (make-cols it (pop rows)))
 (dolist (row rows it) (add it row)))

;;;;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––


(cli)
(with-csv (row (? file)) (print row))
```