```
2. SOURCE CODES
************************************************************
<File project.py>
import lib
import reader
import dist
import tablestr

def project(table):
    d = lib.anyi(len(table.data[0]))
    east, eid = dist.furthest(d, table)
    west, wid = dist.furthest(eid, table)
    x = []
    y = []
    x,y = project0(east, west, table, x, y)
    return widen(table, x, y)


def project0(east, west, table, x, y):
    some = 0.000001 # handles a tedious div/zero error
    # compute the dist between the independent variables
    c = dist.dist(east, west, table.indep, table, False)
    for s in range(len(table.data[0])):
        row = [table.data[k][s] for k in range(len(table.data))]
        a = dist.dist(east, row, table.indep, table, False)
        b = dist.dist(west, row, table.indep, table, False)
        if a > c: x = []; y = []; return project0(east, row, table, x, y)
        if b > c: x = []; y = []; return project0(row, west, table, x, y)
        else:
            temp = (a**2 + c**2 - b**2) / (2*c + some)
            x += [str('%.3f'%temp)]
            y += [str('%.3f'%((a**2 -temp**2)**0.5))]
    return x, y

def widen(table, x, y):
    adds = table.name[:]
    adds += ['$_XX']
    adds += ['$_YY']
    adds += ['$_Hell']
    adds += ['_ZZ']
    ntable = tablestr.Table()
    reader.makeTable(adds, ntable)
    for s in range(len(table.data[0])):
        row = [table.data[k][s] for k in range(len(table.data))]
        tmp = row[:]
        row += [x[s]]
        row += [y[s]]
        row += [str('%.3f'%tablestr.fromHell(tmp, table))]
        row += [str(0)]
        reader.addRow(row, ntable)
    return ntable
import lib
import reader
import dist
import tablestr

def project(table):
    d = lib.anyi(len(table.data[0]))
    east, eid = dist.furthest(d, table)
    west, wid = dist.furthest(eid, table)
    x = []
    y = []
    x,y = project0(east, west, table, x, y)
    return widen(table, x, y)


def project0(east, west, table, x, y):
    some = 0.000001 # handles a tedious div/zero error
    # compute the dist between the independent variables
    c = dist.dist(east, west, table.indep, table, False)
    for s in range(len(table.data[0])):
        row = [table.data[k][s] for k in range(len(table.data))]
        a = dist.dist(east, row, table.indep, table, False)
```



```
        b = dist.dist(west, row, table.indep, table, False)
        if a > c: x = []; y = []; return project0(east, row, table, x, y)
        if b > c: x = []; y = []; return project0(row, west, table, x, y)
        else:
            temp = (a**2 + c**2 - b**2) / (2*c + some)
            x += [str('%.3f'%temp)]
            y += [str('%.3f'%((a**2 -temp**2)**0.5))]
    return x, y

def widen(table, x, y):
    adds = table.name[:]
    adds += ['$_XX']
    adds += ['$_YY']
    adds += ['$_Hell']
    adds += ['_ZZ']
    ntable = tablestr.Table()
    reader.makeTable(adds, ntable)
    for s in range(len(table.data[0])):
        row = [table.data[k][s] for k in range(len(table.data))]
        tmp = row[:]
        row += [x[s]]
        row += [y[s]]
        row += [str('%.3f'%tablestr.fromHell(tmp, table))]
        row += [str(0)]
        reader.addRow(row, ntable)
    return ntable
************************************************************
<File tiles.py>
import project
import math
import tablestr
import reader
import copy
import lib
class Tile:
    def __init__(i):
        i.tiny = 4
        i.big = 0
        i.pre = ''
        i.xs = []
        i.ys = []
        i.header = []
        i.watch = 1
        i.centers = ''

def tiles(table):
    ntable = project.project(table)
    tile = Tile()
    tile.tiny = 4 # the minimum instance num to assign a leaf
    tile.pre = ''
    tile.m = len(table.data[0])  # num of instances have
    tile.big = 2 * math.sqrt(tile.m)
    c1 = 1
    tile.watch = 1
    tile.centers = 'centroids'
    centable = {}  # dictinary to store all the splitted tables including the ce
nter table
    centable0 = tablestr.Table()
    reader.makeTable(ntable.name, centable0)
    centable[0] = centable0
    tiles0(ntable, tile)
    pre = tile.pre
    tiles4(1, tile.m, 1, tile.m, ntable, tile, centable, c1, pre)
    centable['project'] = ntable
    return centable

def tiles0(ntable, tile):
    x = ntable.name.index('$_XX')
    y = ntable.name.index('$_YY')
    #z = ntable.name.index('_ZZ')
    at = []
    for d in range(len(ntable.data[0])):
        tmp = {}
```

```
            tmp['d'] = d
            tmp['x'] = float(ntable.data[x][d])
            tmp['y'] = float(ntable.data[y][d])
            at += [tmp]
    asort(at, 'x', tile)
    asort(at, 'y', tile)

def asort(at, label, tile):
    # func to sort the list based on the label
    att = copy.copy(at)
    while len(att):
        minv = lib.inf; ind = 0
        tmp = {}
        for i in range(len(att)):
            if att[i][label] < minv:
                minv = att[i][label]; ind = i
        tmp = att[ind]
        if label == 'x': tile.xs += [tmp]
        else: tile.ys += [tmp]
        att.pop(ind)

def tiles4(x0, x2, y0, y2, ntable, tile, centable, c1, pre):
    x = x0 + int((x2 - x0)/2)
    y = y0 + int((y2 - y0)/2)
    c1 = tile1(x0, x, y0, y, ntable, tile, centable, c1, pre)
    c1 = tile1(x0, x, y+1, y2, ntable, tile, centable, c1, pre)
    c1 = tile1(x+1, x2, y0, y, ntable, tile, centable, c1, pre)
    c1 = tile1(x+1, x2, y+1, y2, ntable, tile, centable, c1, pre)
    return c1

def tile1(x0, x2, y0, y2, table, tile, centable, c1, pre):
    has = []
    for x in range(x0, x2+1):
        for y in range(y0, y2+1):
            if tile.xs[x-1]['d'] == tile.ys[y-1]['d']:
                has += [tile.xs[x-1]['d']]
    # debug info
    if tile.watch: print '%3s:  '%c1, pre, x0, x2, y0, y2, '#', len(has)
    # recurse: when there is enough data
    if len(has) >= tile.big:
        pre = pre + '|..'
        return tiles4(x0, x2, y0, y2, table, tile, centable,c1, pre)
    # otherwise, new cluster: make a new leaf, only when there is enough data
    if len(has) > tile.tiny:
        # make a new cluster
        makeNewTable(has, c1, table, tile, centable)
        c1 += 1
    #keep track of the num of leaf clusters
    return c1

def makeNewTable(has, c1, table, tile, centable):
    c1 = c1 * 100
    z = table.name.index('_ZZ')
    newtable = tablestr.Table()
    reader.makeTable(table.name, newtable)
    for one in range(len(has)):
        d = has[one]
        row1 = [table.data[s][d] for s in range(len(table.data))]
        row1[z] = str(c1)
        reader.addRow(row1, newtable)
    centers = tablestr.centroid(newtable) #centers[0] is mu or mode
    centers[0][z] = str(c1)
    reader.addRow(centers[0], centable[0])
    centable[c1/100] = newtable
************************************************************
<File lib.py>
import math
import random
inf = 10**17
NINF = -1 * inf
PINCH = 1 / inf
PI = 3.1415926535
EE = 2.7182818284
```



```
def indexes(data):
    rows = []   #get the indexes for the data
    for i in range(len(data)):
        rows.append(i)
    return rows

def rowprint(a, num=6):
    max = len(a)
    line = ''
    for j in range(max):
        line += (a[j] + ',')
    return line

def maybeInt(x):
    return int(x) if x % 1 == 0.0 else float(x)

def norm(x, m, s):
    s += PINCH
    return 1/math.sqrt(2*PI*s**2.0)*EE**(-1*(x-m)**2.0/(2*s**2.0))

def pairs(str):
    tmp = str.split(',')
    lst = {}
    for i in range(len(tmp)/2):
        lst[tmp[2*i]] = tmp[2*i+1]
    return lst

def anyi(n):
    # return a random num within n
    return int(random.random()*n)
************************************************************
<File tablestr.py>
import lib
import math
class Table:
    def __init__(self):
        self.data = []     #data[[col1,...],[col2,...]]
        self.name = []     #name of i-th column
        self.order = []    #order of the col
        self.nump = []     #is i-th column numeric?
        self.wordp = []    #is i-th column non-numeric?
        self.indep = []    #list of indep columns
        self.dep = []      #list of dep columns
        self.less = []     #numeric goal to be minimized
        self.more = []     #numeric goal to be maximized
        self.klass = []    #non-numeric goal
        self.term = []     #non-numeric non-goal
        self.num = []      #numeric non-goal
        # for all cols
        self.n = []        #count of things in this col
        # for wordp columns:
        self.count = []    #count of each word
        self.mode = []     #most common word
        self.most = []     #count of most common word
        # for nump columns:
        self.hi = []       #upper bound
        self.lo = []       #lower bound
        self.mu = []       #mean
        self.m2 = []       #sum of all nums
        self.sd = []       #standard deviation# -*- coding: utf-8 -*-
        # table printing format
        self.CONVFMT = '%4.2f'

def centroid(table):
    "update the mode and most values for wordp type cols or update the mean and
sd values for nump cols"
    rows = [[],[]]
    for c in range(len(table.name)):
        s = table.mode[table.wordp.index(c)] if c in table.wordp else table.CONV
FMT%table.mu[table.nump.index(c)]
        rows[0].append(str(s))
        if table.n[c] == '0':
```

```
            s = 0.0
        else:
            s = float(table.most[table.wordp.index(c)])/table.n[c] if c in table
.wordp else table.sd[table.nump.index(c)]
        rows[1].append(str(table.CONVFMT%s))
    return rows

def tableprint(table, stats=''):
    "print table on the console"
    print ' '
    if stats != '': table.CONVFMT = stats
    print(' ' + lib.rowprint(table.name)+ '  # notes'.ljust(6))
    print('#' + lib.rowprint(centroid(table)[0]) + '  # expected'.ljust(6))
    print('#' + lib.rowprint(centroid(table)[1]) + '  # certainty'.ljust(6))

    for j in range(len(table.data[0])):
        line = []
        for i in range(len(table.data)):
            line.append(table.data[i][j])
        print(' ' + lib.rowprint(line)+ '  #'.ljust(5))

def tableprint_txt(table, f, stats=''):
    "print table on the indicated txt file with table name"
    f.write('\n')
    #f.write('\n' +tablename + '\n'*2)
    if stats != '': table.CONVFMT = stats
    f.write(' ' + lib.rowprint(table.name)+ '  # notes'.ljust(10) + '\n')
    f.write('#' + lib.rowprint(centroid(table)[0]) + '  # expected'.ljust(10) +
'\n')
    f.write('#' + lib.rowprint(centroid(table)[1]) + '  # certainty'.ljust(10) +
 '\n')
    for j in range(len(table.data[0])):
        line = []
        for i in range(len(table.data)):
            line.append(table.data[i][j])
        f.write(' ' + lib.rowprint(line)+ '  #'.ljust(10) + '\n')

def fromHell(row, table):
    m = 0
    out = 0
    for c in table.more:
        if row[c] != '?':
            m += 1
            k = table.nump.index(c)
            out += ((float(row[c]) - float(table.hi[k]))/(float(table.hi[k]) - f
loat(table.lo[k]) + lib.PINCH))**2
    for c in table.less:
        if row[c] != '?':
            m += 1
            k = table.nump.index(c)
            out += ((float(row[c]) - float(table.hi[k]))/(float(table.hi[k]) - f
loat(table.lo[k]) + lib.PINCH))**2
    return math.sqrt(out)/math.sqrt(m) if m else 1
************************************************************
<File dist.py>
import lib

def dist(this, that, lst, table, normF = True):
    sum = 0.0
    for k in lst:
        v1 = this[k]
        v2 = that[k]
        if v1 == '?' and v2 == '?': sum += 1
        elif k in table.nump:
            i = table.nump.index(k)
            mid = (float(table.hi[i]) - float(table.lo[i]))/2
            aLittle = 10**-7
            if v1 == '?':
                if normF: v1 = 1.0 if v2 < mid else 0.0
                else: v1 = table.hi[i] if v2 < mid else table.lo[i]
            else:
                v1 = (float(v1) - float(table.lo[i]))/(float(table.hi[i]) - floa
t(table.lo[i]) + aLittle)
```



```
            if v2 == '?':
                if normF: v2 = 1.0 if v1 < mid else 0.0
                else: v2 = table.hi[i] if v1 < mid else table.lo[i]
            else:
                v2 = (float(v2) - float(table.lo[i]))/(float(table.hi[i]) - floa
t(table.lo[i]) + aLittle)
            sum += (float(v2) -float(v1))**2
        else:
            if v1 == '?': sum += 1.0
            elif v2 == '?': sum += 1.0
            elif v1 != v2: sum +=1.0
            else: sum += 0.0
    return sum**0.5/len(lst)**0.5

def closest(i, table):
    minval = lib.INF
    this = [table.data[s][i] for s in range(len(table.data))]
    for j in range(len(table.data[0])):
        if i == j: continue
        that = [table.data[s][j] for s in range(len(table.data))]
        d = dist(this, that, table.indep, table)
        if d < minval: minval = d; out = j
    row = []
    row = [table.data[s][out] for s in range(len(table.data))]
    return row, out

def furthest(i, table):
    maxval = lib.NINF
    this = [table.data[s][i] for s in range(len(table.data))]
    for j in range(len(table.data[0])):
        if i == j: continue
        that = [table.data[s][j] for s in range(len(table.data))]
        d = dist(this, that, table.indep, table)
        if d > maxval: maxval = d; out = j
    row = []
    row = [table.data[s][out] for s in range(len(table.data))]
    return row, out
************************************************************
<File main.py>
import reader
import tablestr
import tiles
import math
if __name__ == "__main__":
    filename = 'data/nasa93dem.csv'
    table = tablestr.Table()
    reader.readcsv(filename, table)
    tables = tiles.tiles(table)
    ntable = tables['project']
    print '# $_XX'.ljust(8), '$_YY'.ljust(8), 'log(-effort)'.ljust(8)
    for i in range(len(ntable.data[0])):
        print ntable.data[27][i].ljust(8), ntable.data[28][i].ljust(8), str(math.
log(float(ntable.data[24][i]))).ljust(8)
    print '# $_XX'.ljust(8), '$_YY'.ljust(8), 'log($_ZZ)'.ljust(8)
    for i in range(len(tables[0].data[0])):
        print tables[0].data[27][i].ljust(8), tables[0].data[28][i].ljust(8), st
r(math.log(float(tables[0].data[30][i]))).ljust(8)
    for k, v in tables.items():
        print '*'*20
        print 'CLASS LABEL: ', k
        tablestr.tableprint(v)
```