```
#Info for table
#
csvindex = -1 #initialized to -1 as lists start at zero
colname = {k: [] for k in range(1)} #stores dict of names of columns
data = {k: [] for k in range(1)} #stores dict of list of lists containing each r
ow
test = [] #stores test data
#
#metadata
#
order = {k:dict.fromkeys(colname) for k in range(1)} #stores colnames and index
of column in csv
klass = {k: [] for k in range(1)} #dict of list of klass columns
more = {k: [] for k in range(1)} #dict of list of more columns
less = {k: [] for k in range(1)} #dict of list of less columns
num = {k: [] for k in range(1)} #dict of list of num columns
term = {k: [] for k in range(1)} #dict of list of term columns
dep = {k: [] for k in range(1)} #dict of list of dependent columns
indep = {k: [] for k in range(1)} #dict of list of independent columns
nump = {k: [] for k in range(1)} #dict of list containing nump column names
wordp = {k: [] for k in range(1)} #dict of list containing wordp column names
#
#for nump values
#
hi = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing highest va
lues of nump columns
lo = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing lowest val
ues of nump columns
mu = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing mean value
s of nump columns
m2 = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing m2 values
of nump columns
sd = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing std dev of
 nump columns
#
#for wordp values
#
mode = {k:dict.fromkeys(wordp) for k in range(1)} #dictionary containing mode of
 wordp columns
most = {k:dict.fromkeys(wordp) for k in range(1)} #dictionary containing most oc
cured item of wordp columns
count = {k:dict(dict.fromkeys(wordp)) for k in range(1) }#dictionary of dictiona
ries of count of each item in each wordp column
#
#for all
#
n = {k:dict.fromkeys(colname) for k in range(1)} #stores number of elements in e
ach column
isnum = True
#
#for the zeror
#
hypotheses = {}
```

```
import re
from globfile import *
from random import *

def line(csvfile): #returns formatted line from the csvfile
    l = csvfile.readline()
    endcommare = re.compile('.*,$')
    if l ≠ '':
        l = l.split('#')[0]
        l = l.replace('\t','')
        l = l.replace('\n','')
        l = l.replace(' ','')
        endcomma = endcommare.match(l)
        if endcomma:
            return l+line(csvfile)
        else:
            return l
    else:
        return -1

def rowprint(row): #returns neat rows
    columns = [ "%15s" % cell for cell in row]
    columns.append("%4s" % '#')
    return ' '.join(columns)

def expected(row,z): #returns expected outcome
    out = [c for c in colname[z]]
    for c in row:
        if c in wordp[z]:
            out[colname[z].index(c)] = str(mode[z][c])
        else:
            out[colname[z].index(c)] = str('%0.2f' % round(mu[z][c],2))
    return out

def indexes(lst):
    out = []*len(lst)
    for i in lst:
        out[i] = i
    return out

def newdlist(name, key):
    name[key] = []

def newddict(name,key):
    name[key] = {}

def newddictdict(name,key,c):
    name[key][c] = {}

def indexes(data,z):
    return data[z]

def shuffled(rows):
    shuffle(rows)
```

```
from reader import *
from table import *
from sys import argv
from xval import *

csvfile = open('../data/weather1.csv','r')
readCsv(csvfile,argv[1]) #takes predicted value as arguement
print data[argv[1]]
xvals(data,2,3,'zeror',argv[1])

#tableprint(argv[1])
```

```
import re
from lib import *

def makeTable(str,z):
    newdlist(klass,z)
    newddict(order,z)
    newdlist(less,z)
    newdlist(num,z)
    newdlist(term,z)
    newdlist(dep,z)
    newdlist(indep,z)
    newdlist(nump,z)
    newdlist(wordp,z)
    newdlist(colname,z)
    newdlist(data,z)
    newddict(count,z)
    newddict(n,z)
    newddict(mode,z)
    newddict(most,z)
    newddict(hi,z)
    newddict(lo,z)
    newddict(mu,z)
    newddict(m2,z)
    newddict(sd,z)
    newdlist(data,z)

    csvindex = -1
    for csvcol in str:
        isnum=True
        csvindex+=1
        ignore = re.match('\?.*$',csvcol)
        if ignore:
            continue
        else:
            colname[z].append(csvcol)
            order[z][csvcol] = csvindex
            klasschk = re.match('=.*$',csvcol)
            morechk = re.match('\+.*$',csvcol)
            lesschk = re.match('-.*$',csvcol)
            numchk = re.match('\$.*$',csvcol)
            if klasschk:
                dep[z].append(csvcol)
                klass[z].append(csvcol)
                isnum = False
            elif morechk:
                dep[z].append(csvcol)
                more[z].append(csvcol)
            elif lesschk:
                dep[z].append(csvcol)
                less[z].append(csvcol)
            elif numchk:
                indep[z].append(csvcol)
                num[z].append(csvcol)
            else:
                indep[z].append(csvcol)
                term[z].append(csvcol)
                isnum = False
            n[z][csvcol] = 0
            if isnum:
                nump[z].append(csvcol)
                hi[z][csvcol] = 0.1 * (-10**13)
                lo[z][csvcol] = 0.1 * (10**13)
                mu[z][csvcol] = 0.0
                m2[z][csvcol] = 0.0
                sd[z][csvcol] = 0.0
            else:
                wordp[z].append(csvcol)
                count[z][csvcol] = {}
                mode[z][csvcol] = 0
                most[z][csvcol] = 0

def addRow(lst,z):
    temp = []
```



```
    skip = False
    for c in klass[z]:
        csvindex = order[z][c]
        item = lst[csvindex]
        if item ≠ z:
            skip = True
        if z ≡ "zero":
            skip = False
    for c in colname[z]:
        csvindex = order[z][c]
        item = lst[csvindex]
        uncertain = re.match('\?',str(item))
        if skip:
            return
        if uncertain:
            temp.append(item)
        else:
            n[z][c] += 1
            if c in wordp[z]:
                temp.append(item)
                try:
                    new = count[z][c][item] = count[z][c][item] + 1
                    if new > most[z][c]:
                        most[z][c] = new
                        mode[z][c] = item
                except KeyError:
                    count[z][c][item] = 1
                    if count[z][c][item] > most[z][c]:
                        most[z][c] = 1
                        mode[z][c] = item
            else:
                item = float(item)
                temp.append(item)
                if item > hi[z][c]:
                    hi[z][c] = item
                if item < lo[z][c]:
                    lo[z][c] = item
                delta = item - mu[z][c]
                mu[z][c] += delta / n[z][c]
                m2[z][c] += delta * (item - mu[z][c])
                if n[z][c] > 1:
                    sd[z][c] = (m2[z][c] / (n[z][c] - 1))**0.5
    data[z].append(temp)

def readCsv(csvfile,z):
    seen = False
    FS = ','
    while True:
        str = line(csvfile)
        if str ≡ -1:
            print 'WARNING: empty or missing file'
            return -1
        str = str.split(FS)
        if str ≠ ['']:
            if seen:
                addRow(str,z)
            else:
                seen = True
                makeTable(str,z)
```

```
from globfile import *
from lib import *
def tableprint(z): #prints table with the summary
    print rowprint(colname[z]),'%10s' % 'notes'
    print rowprint(expected(colname[z],z)), '%10s' % 'expected'
    temp = [ c for c in range(len(colname[z]))]
    for c in colname[z]:
        if c in nump[z]:
            temp[colname[z].index(c)] = str('%0.2f' % round(sd[z][c],2))
        else:
            temp[colname[z].index(c)] = str('%0.2f' % round(float(most[z][c])/fl
oat(n[z][c]),2))
    print rowprint(temp),'%10s' % 'certainity'
    for row in data[z]:
        print rowprint(row)

def klass1(data, z):
    for k in klass[z]:
        return data[colname[z].index(k)]

def klassAt(z):
    for k in klass[z]:
        return order[z][k]
```

```
#! /bin/python
from lib import *
from reader import *
from table import *
from zeror import *

def xvals(data,x,b,f,z):
    rows = indexes(data,z)
    s = int(len(rows)/b)
    while x>0:
        shuffled(rows)
        for b1 in range(0,b):
            xval(b1*s+1, (b1+1)*s, data, rows, f, z)

def  xval(start, stop, data, rows, f, z):
    rmax = len(rows)
    test = []
    for r in range(1, rmax):
        d = rows[r]
        if r ≥ start ∧ r ≤ stop:
            test.append(d)
        else:
            temp = klass1(d, z)
            try:
                hypotheses[temp] += 1
                if hypotheses[temp] ≡ 1:
                    makeTable(colname[temp],temp)
                addRow(d,z)
            except KeyError:
                hypotheses[temp] = 0
    zeror(test, data, hypotheses, z)
    xvalTest1(test,data,hypotheses)

def xvalTest1(test,data,hypotheses):
    print "\n================================="
    for h in hypotheses:
        tableprint(h)
```

```
from reader import *
from  xval import *
from lib import *

def zeror(test,data,hypotheses,z):
    hmost = -10**23
    acc = 0
    for h in hypotheses:
        these = len(data[h])
        if these > hmost:
            most = these
            got = h
    print "#got: ",got
    where = klassAt(z)
    print test,'testttttt'
    print data,'dataaaaaaaaaa'
    for t in test:
        want = data[z][data[z].index(t)][where]
        if want == got:
            acc+=1
    print 100*acc/len(test),"\t"
```