```
from pandas import DataFrame


class Evaluator:

    def __init__(self, m, inf):
        """ This class is used to evaluate an information retrieval
        system based on Relevant / Non-relevant annotations and general
        information concerning the document, check main() for examples. """
        self.m = m
        self.inf = inf
        self.qd = self.rn_matrix(m)

    def frame_ord(self, data, r, c):
        """ Frames almost all matrices in this evaluator. """
        return DataFrame(data, index=r, columns=c)

    def frame_mat(self, data, c):
        """ Used to frame the qrank. """
        return DataFrame(data, columns=c)

    def rn_matrix(self, m):
        """ The RN matrix returns tuples per Query in the
        form of (Relevant, Non-relevant). """
        qd = {}
        for i in range(0, len(m)):
            rel, nrel = 0.0, 0.0
            for j in range(0, len(m[i])):
                if m[i][j] == 'R':
                    rel += 1
                else:
                    nrel += 1
            qd['q'+str(i+1)] = (rel, nrel)
        return qd

    def conf_matrix(self, i):
        """ This is a basic confusion matrix with false/
        true postives/negatives based on the actual relevance
        and the retrieved relevance. """
        tp = self.qd[i][0]
        fp = self.qd[i][1]
        fn = self.inf[i] - self.qd[i][0]
        tn = self.inf['tot'] - tp - fp - fn
        return {'tp': tp,
                'fp': fp,
                'fn': fn,
                'tn': tn}

    def precision(self, i):
        m = self.conf_matrix(i)
        return m['tp'] / (m['tp'] + m['fp'])

    def recall(self, i):
        m = self.conf_matrix(i)
        return m['tp'] / (m['tp'] + m['fn'])

    def f_measure(self, beta, i):
        P, R = self.precision(i), self.recall(i)
        return ((beta**2+1)*P*R)/(beta**2*P+R)

    def accuracy(self, i):
        m = self.conf_matrix(i)
        return (m['tp'] + m['tn']) / (m['tp'] + m['tn'] + m['fp'] + m['fn'])

    def qrank(self, i, k, p=None):
        """ Goes down the query retrieved R/N and calculates
        their precision (relv/float(x+1) and recall, as well
        as ranks them. """
        ii, qr = self.m[int(i.replace('q', ''))-1], []
        r, relv, ri = 0, 0, 1.00/float(self.inf[i])
        for x in range(0, k):
            rsw = ''
            if ii[x] == 'R':
                r += ri; relv += 1; rsw = 'X'
            qr.append([x+1, rsw, r, relv/float(x+1)])
        if p: print(self.frame_mat(qr, ['rank', 'rel', 'R', 'P']))
```

```
        return qr

    def map(self, k, p=None):
        """ Grabs only relevant averages from qrank. """
        tl = []
        for i in range(0, len(self.m)):
            m, tot, c = self.qrank('q'+str(i+1), k, p), 0, 0
            for j in range(0, len(m)):
                if m[j][1] == 'X':
                    tot += m[j][3]; c += 1
            try:
                tl.append(tot/c)
            except ZeroDivisionError:
                tl.append(0.0)
        return sum(tl)/len(self.m)

    def kmeasure(self, m2):
        """ The kmeasure simply matricifies the agreements
        between annotator X and Y. """
        m, rr, nn, rn, nr = self.m, 0, 0, 0, 0
        for i in range(0, len(self.m)):
            for j in range(0, len(self.m[i])):
                if m[i][j] == 'R' and m2[i][j] == 'R':
                    rr += 1
                elif m[i][j] == 'N' and m2[i][j] == 'N':
                    nn += 1
                elif m[i][j] == 'R' and m2[i][j] == 'N':
                    rn += 1
                elif m[i][j] == 'N' and m2[i][j] == 'R':
                    nr += 1
        return {'rr': float(rr), 'nn': float(nn), 'rn': float(rn), 'nr': float(nr)}

    def kappa(self, m2):
        """ Kappa grabs pa - pe / 1 - pe based on agreement. """
        km = self.kmeasure(m2)
        pa = (km['rr'] + km['nn']) / sum(km.values())
        pn = (km['nr'] + km['nn'] + km['rn'] + km['nn']) / (sum(km.values()) * 2)
        pr = (km['rr'] + km['rn'] + km['rr'] + km['nr']) / (sum(km.values()) * 2)
        pe = pn**2 + pr**2
        return (pa - pe) / (1 - pe)


def main():
    #-------------------------------------------------------------

    #initial annotator
    M  = [['R', 'N', 'R', 'N', 'R', 'R', 'N', 'N', 'R', 'N'],  # q1
          ['R', 'R', 'N', 'N', 'R', 'N', 'N', 'R', 'N', 'N'],  # q2
          ['N', 'R', 'R', 'R', 'R', 'N', 'N', 'N', 'R', 'N'],  # q3
          ['R', 'N', 'R', 'N', 'R', 'N', 'R', 'N', 'N', 'N']]  # q4
    #second annotator
    M2 = [['R', 'R', 'R', 'R', 'R', 'R', 'N', 'N', 'N', 'N'],  # q1
          ['R', 'R', 'R', 'N', 'R', 'N', 'N', 'N', 'N', 'N'],  # q2
          ['N', 'R', 'R', 'R', 'R', 'N', 'N', 'N', 'R', 'N'],  # q3
          ['R', 'N', 'R', 'N', 'R', 'N', 'N', 'N', 'N', 'N']]  # q4
    #general information
    inf = {'tot': 250, 'q1': 10, 'q2': 12, 'q3': 15, 'q4': 8}

    #-------------------------------------------------------------
    ev = Evaluator(M, inf)
    res, conf, tab = [], [], []

    for i in range(0, len(M)):
        q = 'q'+str(i+1)
        res.append([ev.precision(q),
                    ev.recall(q),
                    ev.f_measure(1.0, q),
                    ev.accuracy(q)])
        conf.append(ev.conf_matrix(q).values())

    eva = [ev.map(1),
           ev.map(3),
           ev.map(5),
           ev.map(10),
           ev.kappa(M2)]

    #only for debugging, produces crap
```

```
    ev.map(1, 'print')
    ev.map(3, 'print')
    ev.map(5, 'print')
    ev.map(10, 'print')

    print(ev.frame_ord(conf,
                       ['q1', 'q2', 'q3', 'q4'],
                       ['fp', 'tn', 'fn', 'tp']), '\n')
    print(ev.frame_ord([[round(y, 3) for y in x] for x in res],
                       ['q1', 'q2', 'q3', 'q4'],
                       ['P', 'R', 'F', 'A']))
    print(ev.frame_ord([round(y, 3) for y in eva],
                       ['map(1)', 'map(3)', 'map(5)', 'map(10)', 'kappa'],
                       ['']))

if __name__ == '__main__':
    main()
```

```
1     2      0.125  0.500000
2     3   X  0.250  0.666667
   rank rel    R         P
0     1   X  0.1  1.000000
1     2      0.1  0.500000
2     3   X  0.2  0.666667
3     4      0.2  0.500000
4     5   X  0.3  0.600000
   rank rel         R         P
0     1   X  0.083333  1.000000
1     2   X  0.166667  1.000000
2     3      0.166667  0.666667
3     4      0.166667  0.500000
4     5   X  0.250000  0.600000
   rank rel         R         P
0     1      0.000000  0.000000
1     2   X  0.066667  0.500000
2     3   X  0.133333  0.666667
3     4   X  0.200000  0.750000
4     5   X  0.266667  0.800000
   rank rel      R         P
0     1   X  0.125  1.000000
1     2      0.125  0.500000
2     3   X  0.250  0.666667
3     4      0.250  0.500000
4     5   X  0.375  0.600000
   rank rel    R         P
0     1   X  0.1  1.000000
1     2      0.1  0.500000
2     3   X  0.2  0.666667
3     4      0.2  0.500000
4     5   X  0.3  0.600000
5     6   X  0.4  0.666667
6     7      0.4  0.571429
7     8      0.4  0.500000
8     9   X  0.5  0.555556
9    10      0.5  0.500000
   rank rel         R         P
0     1   X  0.083333  1.000000
1     2   X  0.166667  1.000000
2     3      0.166667  0.666667
3     4      0.166667  0.500000
4     5   X  0.250000  0.600000
5     6      0.250000  0.500000
6     7      0.250000  0.428571
7     8   X  0.333333  0.500000
8     9      0.333333  0.444444
9    10      0.333333  0.400000
   rank rel         R         P
0     1      0.000000  0.000000
1     2   X  0.066667  0.500000
2     3   X  0.133333  0.666667
3     4   X  0.200000  0.750000
4     5   X  0.266667  0.800000
5     6      0.266667  0.666667
6     7      0.266667  0.571429
7     8      0.266667  0.500000
8     9   X  0.333333  0.555556
9    10      0.333333  0.500000
```

✓ 0s completed at 11:51 AM