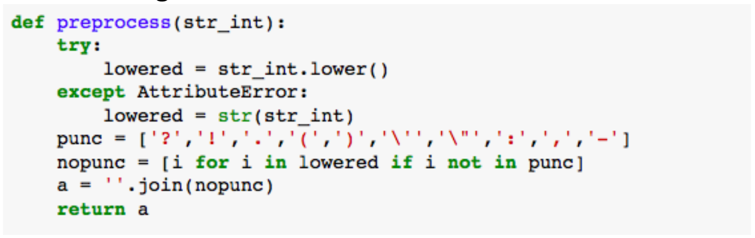
Homework 1

Preprocessing:

For the Preprocessing problem I developed a program that will remove specific punctuation from a string. This same format will later be used for removing stop words:

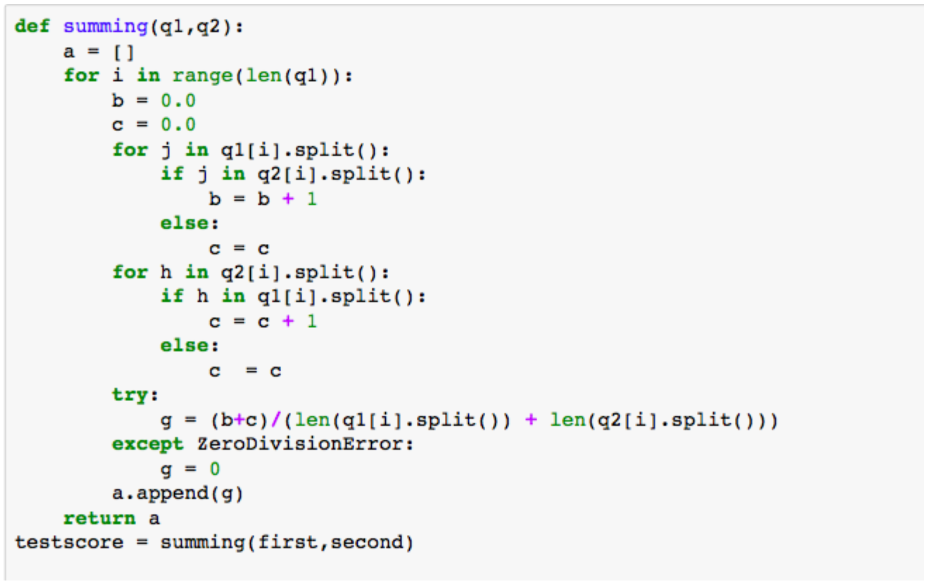


```python
def preprocess(str_int):
    try:
        lowered = str_int.lower()
    except AttributeError:
        lowered = str(str_int)
    punc = ['?','!','.','(',')','\'','\"',':',',','-']
    nopunc = [i for i in lowered if i not in punc]
    a = ''.join(nopunc)
    return a
```

We use a generator loop to get rid of the punctuation and then join all of the strings together.

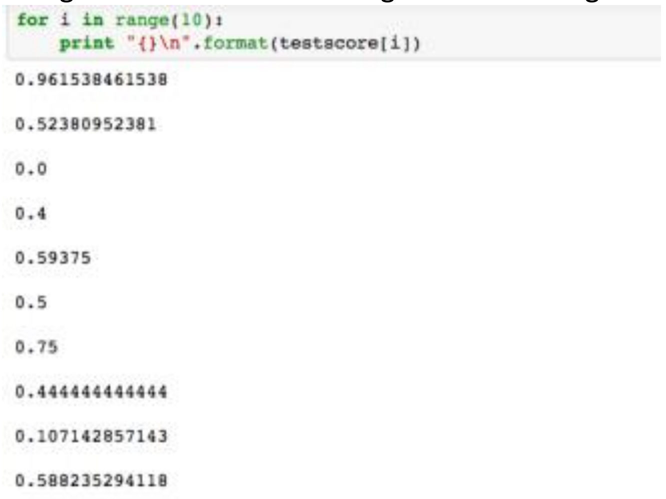
Part 2:

After Processing the data we use the function below to obtain the scores. In this problem we use our function to count if one question’s words are in the other and then count them. We then obtain a percentage of words in the first list that are in the second list and second list in the first list.



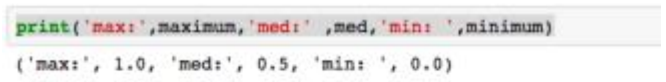
```python
def summing(q1,q2):
    a = []
    for i in range(len(q1)):
        b = 0.0
        c = 0.0
        for j in q1[i].split():
            if j in q2[i].split():
                b = b + 1
            else:
                c = c
        for h in q2[i].split():
            if h in q1[i].split():
                c = c + 1
            else:
                c  = c
        try:
            g = (b+c)/(len(q1[i].split()) + len(q2[i].split()))
        except ZeroDivisionError:
            g = 0
        a.append(g)
    return a
testscore = summing(first,second)
```

Using the above function we get the following results.



```
for i in range(10):
    print "{}\n".format(testscore[i])
```

```
0.961538461538

0.52380952381

0.0

0.4

0.59375

0.5

0.75

0.444444444444

0.107142857143

0.588235294118
```



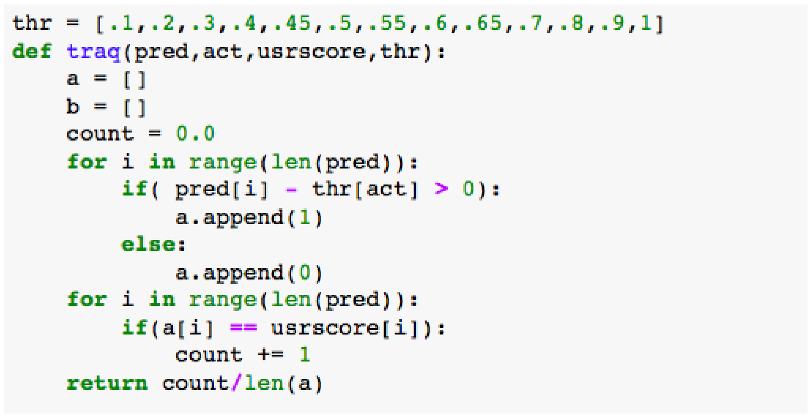
```
print('max:',maximum,'med:' ,med,'min: ',minimum)
```

```
('max:', 1.0, 'med:', 0.5, 'min: ', 0.0)
```

From this report we see that the statistics are well distributed from 1 to 0. .5 is the median suggesting that the distribution centers around .5 and distributed evenly from 1 to 0.

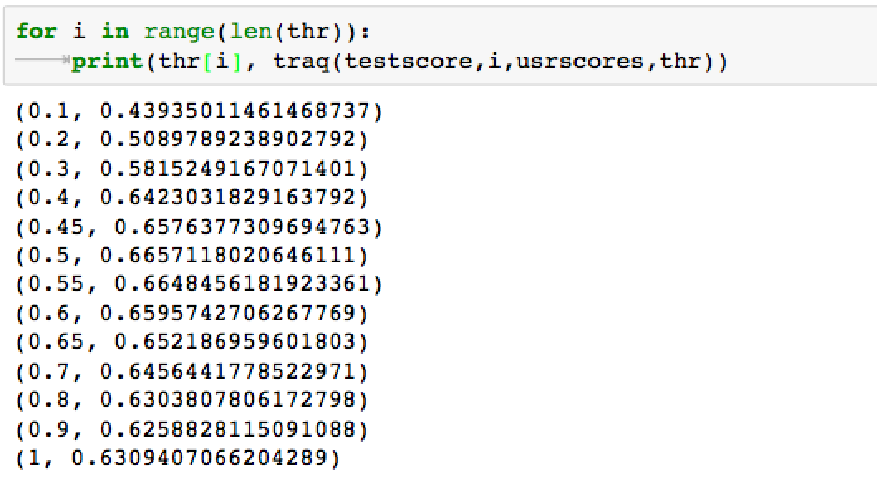
Part 3:

Now we look at the accuracy scores on predicting the validation.csv. Using the function traq I changed the scores by subtracting a certain threshold. Once I implemented the threshold I then converted numbers to one if they were greater than 0 and 0 if they were less than 0.



```
thr = [.1,.2,.3,.4,.45,.5,.55,.6,.65,.7,.8,.9,1]
def traq(pred,act,usrscore,thr):
    a = []
    b = []
    count = 0.0
    for i in range(len(pred)):
        if( pred[i] - thr[act] > 0):
            a.append(1)
        else:
            a.append(0)
    for i in range(len(pred)):
        if(a[i] == usrscore[i]):
            count += 1
    return count/len(a)
```

We then look at the output using a for loop to observe which threshold worked best.



```
for i in range(len(thr)):
    print(thr[i], traq(testscore,i,usrscores,thr))
```

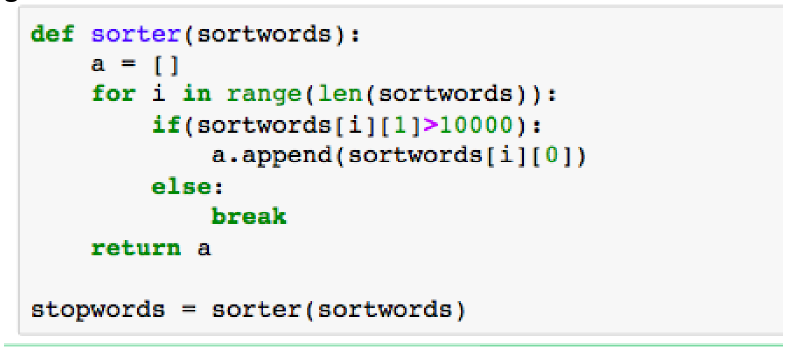
```
(0.1, 0.43935011461468737)
(0.2, 0.5089789238902792)
(0.3, 0.5815249167071401)
(0.4, 0.6423031829163792)
(0.45, 0.6576377309694763)
(0.5, 0.665711802064611)
(0.55, 0.6648456181923361)
(0.6, 0.6595742706267769)
(0.65, 0.652186959601803)
(0.7, 0.6456441778522971)
(0.8, 0.6303807806172798)
(0.9, 0.6258828115091088)
(1, 0.6309407066204289)
```

From the thresholding we see a peak at .5 This makes sense as a threshold because it is making all of the words in one list that were at least half in the other list means would be 1 and the others would be 0.

Part 4:
In part four I used a counting function to find the count of each word. By using the counter I could then sort

```
from collections import Counter
newlist = [list(df['q1']),list(df['q2'])]
onelist = sum(newlist,[])
ourlist = ', '.join(onelist)
finlist = preprocess(ourlist)
wordcount = Counter(finlist.split()
sortwords = sorted(wordcount.items(), key = lambda x:x[1], reverse = True)
```

Once I counted the words I then sorted them. This led to me then finding each word that was greater than 10000.



```
def sorter(sortwords):
    a = []
    for i in range(len(sortwords)):
        if(sortwords[i][1]>10000):
            a.append(sortwords[i][0])
        else:
            break
    return a

stopwords = sorter(sortwords)
```

After this I used the preprocess function but I swapped the punctuation with the stop words. After than I then ran the part three functions again.

```python
def stoptheword(str_int,stopwords):
    try:
        lowered = str_int.lower()
    except AttributeError:
        lowered = str(str_int)

    nopunc = [i for i in lowered.split() if i not in stopwords]
    a = ' '.join(nopunc)
    return a
```

The Valid function in the next part combines all the previous functions I wrote to measure the accuracy.

```python
def valid(testdata,thr,userscore,stopwords,act):
    dfs = dfmaker(testdata)
    validf = qmaker(dfs[0],dfs[1])
    stopdf = qmakertwo(validf['q1'],validf['q2'],stopwords)
    stopfirst = list(stopdf['q1'])
    stopsecond = list(stopdf['q2'])
    findf = summing(stopfirst,stopsecond)
    scores = traq(findf,act,userscore,thr)
    return scores
```

When looking at the comparing scores we don't see a unimodal scoring anymore. Instead the values go up until threshold .5 and then go down, but then fluctuate from .65 onwards. In this part. .65 is the maximum threshold; however, the results are only slightly improved from no stop words.

```python
for i in range(len(thr)):
    print(thr[i], valid(testdata,thr,userscore,stopwords,i))
```

```
(0.1, 0.4929904235363807)
(0.2, 0.5473516635403297)
(0.3, 0.609549313851318)
(0.4, 0.6548030407740152)
(0.45, 0.6643671635896929)
(0.5, 0.6658480600256689)
(0.55, 0.664737387698687)
(0.6, 0.6606896041070195)
(0.65, 0.658974232402014)
(0.7, 0.6540996149669267)
(0.8, 0.6534578931780037)
(0.9, 0.6615164379504394)
(1, 0.6302448415440813)
```