Q1

I split the training data into two lists by importing **train_test_split** from **sklearn.model_selection**. By setting the parameters to **train_size=0.8**, **test_size=0.2**, **shuffle=False**, **stratify=None,** I made sure that I get the first 80% for training and the remaining 20% for testing. Then I trained the tagger using only the 80% split. After that I tested the 20% split and printed out the classification report and confusion matrix. The macro average for the f1-score at this point is **0.55**.

Q2

By observing the classification report printed out from question 1, the 5 classes with the **lowest precision** are **B-Opinion, B-Plot, B-Soundtrack, I-Opinion, I-Soundtrack**. To find all the sentences where there is a false positive, I created a list of categories including the 5 classes with the lowest precision, then created a "for loop" to catch all the sentences where the predicted label is different than the true label and listed in the categories list that I created including the 5 classes:

```
for i in range(len(sent)):
    if sent_preds[i] != sent_true[i]:
      if sent_preds[i] in categories:
```

To make it readable and presentable, I used the **PrettyTable** library to output the words, true label, and predicted label for each sentence. Finally to remove duplicated sentences which might have 2 or more words predicted wrong, I used **np.unique** and printed out all the sentences where there is a false positive.

Q3

By observing the classification report printed out from question 1, the 5 classes with the **lowest recall** are **B-Character_Name, B-Soundtrack, I-Character_Name, I-Opinion, I-Soundtrack**. To find all the sentences where there is a false negative, I created a list of categories including the 5 classes with the lowest recall and used the same procedure as question 2. The only difference is that the "for loop" is catching the true label that is listed in the categories list, rather than the predicted label **(if sent_true2[i] in categories2:)**. Meaning that it gets the sentences where the true label is included in those 5 classes, but not predicted correctly **(if sent_preds2[i] != sent_true2[i]:)**. Then I printed out the sentences using **PrettyTable** and removed the duplicate sentences by using **np.unique**.

Q4

I altered the **preProcess** function to concatenate the word and the POS tag together using the special character **"@"** in the middle, and then I pre-processed the 80% training data. After that, I created a "for loop" to catch all the sentences in the pre-processed 80% data. Then, using the **get_features** function, I modified the function to split on the special symbol **"@"** using **token = token.split('@')** and appended in the **feature_list** the **"POS_" + token[1]** so that it gets the POS tag. After modifying the **get_features** function, I trained the tagger using the pre-processed 80% training data and then used the same "for loop" for the 20% test data to test the tagger. After printing out the classification report I've noticed that the macro average f1-score increased to **0.56**.

Q5

In order to get the best results for the macro average f1-score, I experimented with different features by adjusting the **get_features** function. Firstly, I added more suffixes up to length 6, and then I added prefixes for up to length 6 using:

```
if len(token[0]) > 1:
    feature_list.append("PRE_" + token[0][:-1])
```

After adding more suffixes and prefixes, I created conditions to get the previous and the next words. Because the first word of a sentence cannot have a previous word, and the last word of a sentence cannot have a next word, I used the following lines of code to prevent errors:

For previous word:

```
if idx == 0:
    print()
else:
    feature_list.append("PREV_WORD_" + tokens[idx-1].split('@')[0])
```

For next word:

```
if idx == len(tokens)-1:
    print()
  else:
    feature_list.append("NEXT_WORD_" + tokens[idx+1].split('@')[0])
```

After modifying the **get_features** function, I trained the tagger using the 80% training data and added **training_opt={“feature.minfreq”:2}** to the parameters of **CRFTagger** and then tested the tagger using the 20% test data to get the classification report as well as the confusion matrix. The macro average f1-score of the classification now is **0.65**. I tried tweaking the parameters of the **CRFTagger** to see if I could get even better results by adding **c1**, **c2** regularization parameters, and other **training_opt** options, but led to decreasing the macro average f1-score rather than increasing it. Finally, to test the original test data, I firstly got the raw training data from **"trivia10k13train.bio.txt"** and then pre-processed it using the **preProcess** function that I created earlier. And then I did the same for the raw test data in **"trivia10k13test.bio.txt"**. After that, I trained the training data using the same **get_features** function and **CRFTagger** parameters that got me the best results when I tested the 20% test data and then tested the tagger on the original test data and finished off by printing the classification report with the macro average f1-score being **0.61**.