# Q2.4: Analyze options of tokenize function:

Table 1: Observations

| | Lemmatized | No_stopwords | Recall |
|---|---|---|---|
| #1 | No | No | 0.6847826086956522 |
| #2 | Yes | No | 0.8043478260869565 |
| #3 | No | Yes | 0.7119565217391305 |
| #4 | Yes | Yes | 0.7717391304347826 |

- As we can see, using lemmatized tokens gives higher recall of ~80% and ~77% respectively. This tells us that lemmatization helps a lot in improving performance.
- We got the worst recall of ~68%, when both the options are disabled i.e. lemmatized: no and no_stopwords: no. This tells us the importance of doing mentioned processing of tokens before using them as features.
- Also, if we look at overall performance improvement then enabling lemmatized helps more than enabling no_stopwords option

# Q3.2:

Table 2: Observations [for ref.]

| | threshold | Lemmatized | No_stopwords | Precision | Recall |
|---|---|---|---|---|---|
| #1 | 0.1 | No | No | 0.4134831460674157 | 1.0 |
| | | Yes | No | 0.4017467248908297 | 1.0 |
| | | No | Yes | 0.4265734265734265 | 0.99456521739130 |
| | | Yes | Yes | 0.40979955456570155 | 1.0 |
| #2 | 0.2 | No | No | 0.4412470023980815 | 1.0 |
| | | Yes | No | 0.4191343963553531 | 1.0 |
| | | No | Yes | 0.44938271604938274 | 0.98913043478260 |
| | | Yes | Yes | 0.42790697674418604 | 1.0 |
| #3 | 0.3 | No | No | 0.4861878453038674 | 0.95652173913043 |
| | | Yes | No | 0.45112781954887216 | 0.97826086956521 |
| | | No | Yes | 0.473972602739726 | 0.94021739130434 |
| | | Yes | Yes | 0.4475 | 0.97282608695652 |
| #4 | 0.4 | No | No | 0.5187713310580204 | 0.82608695652173 |
| | | **Yes** | **No** | **0.48830409356725146** | **0.90760869565217** |
| | | No | Yes | 0.50814332247557 | 0.84782608695652 |
| | | Yes | Yes | 0.4699140401146132 | 0.89130434782608 |
| #5 | 0.5 | No | No | 0.5550660792951542 | 0.68478260869565 |
| | | Yes | No | 0.5229681978798587 | 0.80434782608695 |
| | | No | Yes | 0.532520325203252 | 0.71195652173913 |
| | | Yes | Yes | 0.4797297297297297 | 0.77173913043478 |
| #6 | 0.6 | No | No | 0.5491329479768786 | 0.51630434782608 |
| | | Yes | No | 0.5258215962441315 | 0.60869565217391 |
| | | No | Yes | 0.5409836065573771 | 0.53804347826086 |
| | | Yes | Yes | 0.519650655021834 | 0.64673913043478 |
| #7 | 0.7 | No | No | 0.5508474576271186 | 0.35326086956521 |
| | | Yes | No | 0.5369127516778524 | 0.43478260869565 |
| | | No | Yes | 0.5985915492957746 | 0.46195652173913 |
| | | Yes | Yes | 0.5666666666666667 | 0.55434782608695 |
| #8 | 0.8 | No | No | 0.5846153846153846 | 0.20652173913043 |
| | | Yes | No | 0.5862068965517241 | 0.27717391304347 |
| | | No | Yes | 0.6341463414634146 | 0.28260869565217 |
| | | Yes | Yes | 0.6160714285714286 | 0.375 |
| #9 | 0.9 | No | No | 0.8 | 0.10869565217391 |
| | | Yes | No | 0.7647058823529411 | 0.14130434782608 |
| | | No | Yes | 0.6888888888888889 | 0.16847826086956 |
| | | Yes | Yes | 0.6101694915254238 | 0.19565217391304 |

- From table 2, we can observe that as threshold is changing from 0.1 to 0.9, recall is mostly decreasing and precision is mostly fluctuating between 0.4 to 0.6.
- **For this specific problem,** to get the best performance, I would choose parameters as follows(based on the observations from table 2):
  - Threshold: 0.4, Lemmatized: Yes, no_stopwords: No
- I believe duplicates which have tokens with similar roots/lemmas can be found easily using **this approach**. But duplicates which have tokens **with different roots/lemmas but which are synonyms** can be difficult to find using this approach.
  - Example: For below questions, we need more robust approach.
    - Q1: - Can I install Ubuntu and Windows side by side?
    - Q2: - How do I dual boot Windows alongside Ubuntu?
- Yes, utilizing TF-IDF values as features in similarity finding approaches can be quite useful.