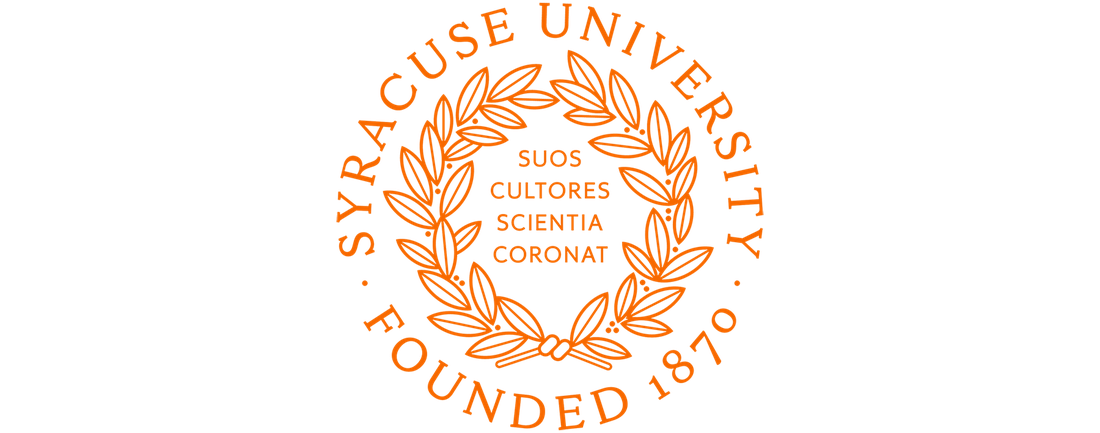
**ChihHao Luca Yuan**

**Syracuse University**

**NLP Final Project**

**Kaggle competition movie review**



**Contents**

[**1. Setting 2**](#_Toc184592812)

[**2. Analysis 4**](#_Toc184592813)

[**a. Without any processing. 4**](#_Toc184592814)

[**b. Remove stop words 6**](#_Toc184592815)

[**c. Lemmatization → Removing Stopwords → Bag-of-Words. 7**](#_Toc184592816)

[**d. Using negation features. 8**](#_Toc184592817)

[**e. Subjectively: SL 10**](#_Toc184592818)

[**f. Bi-grams into Feature Extraction 11**](#_Toc184592819)

[**g. POS Tags Followed by Bi-grams 12**](#_Toc184592820)

[**h. POS Tag Features 14**](#_Toc184592821)

[**i. LIWC 15**](#_Toc184592822)

[**j. Combine bigram and LIWC 16**](#_Toc184592823)

[**k. Sklearn RandomForestClassifier 17**](#_Toc184592824)

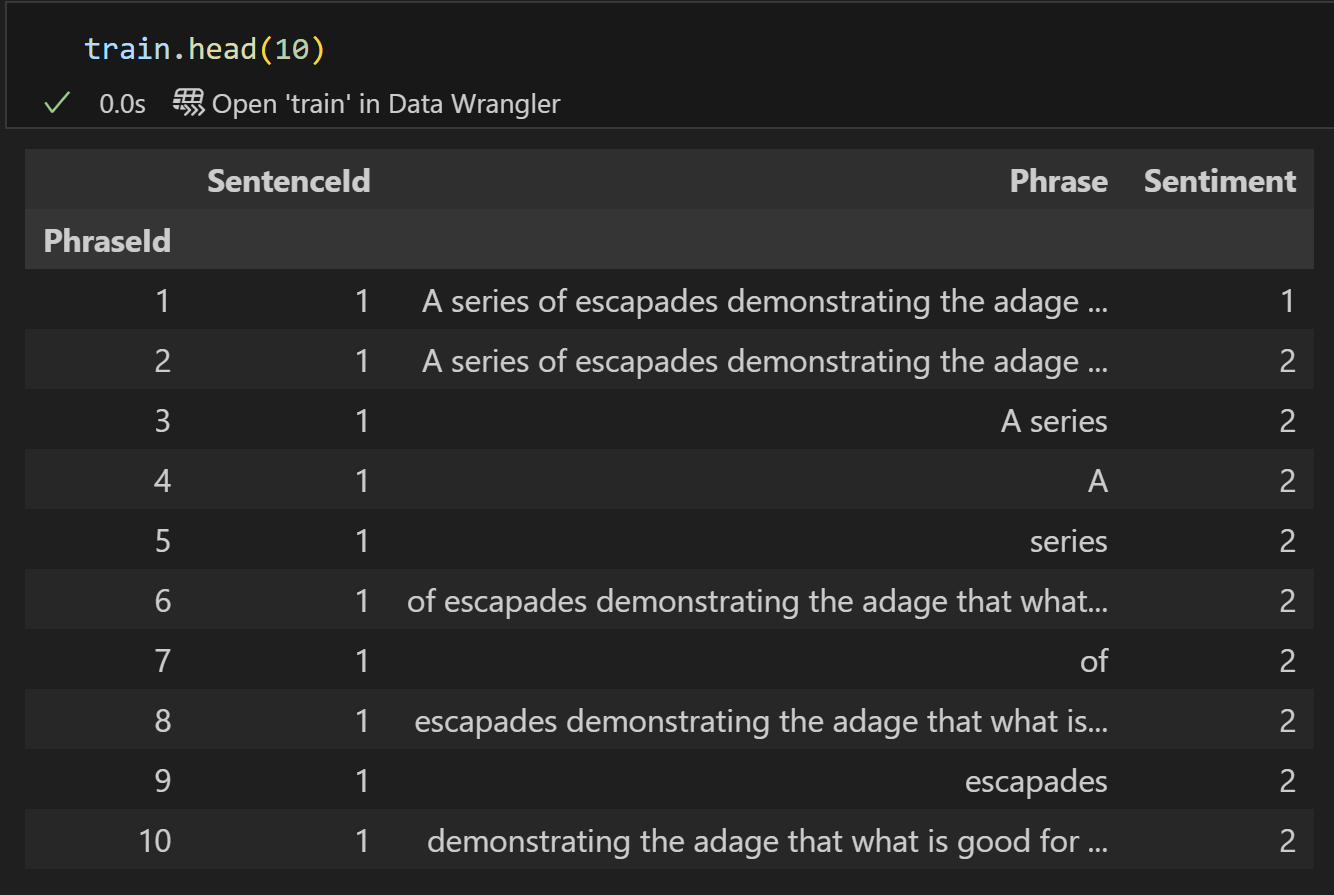
[**3. Conclusion 19**](#_Toc184592825)

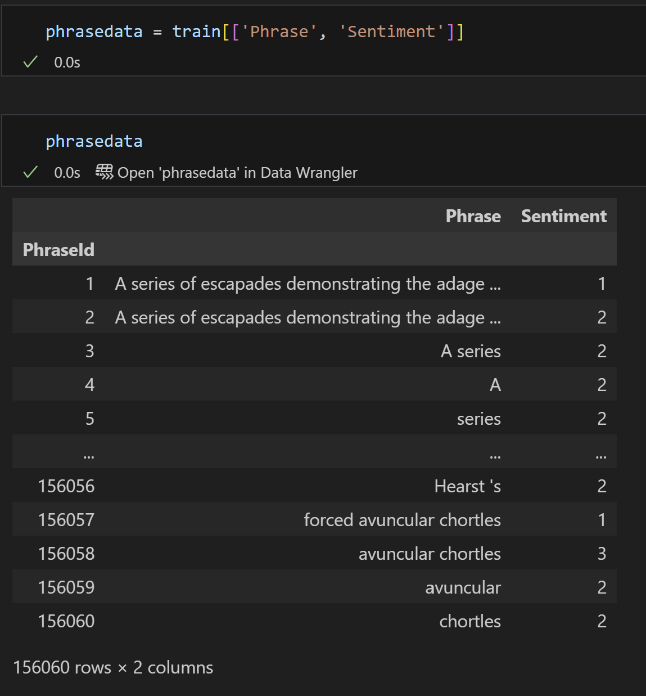
1. **Setting**

Set up Libraries and Tools Used

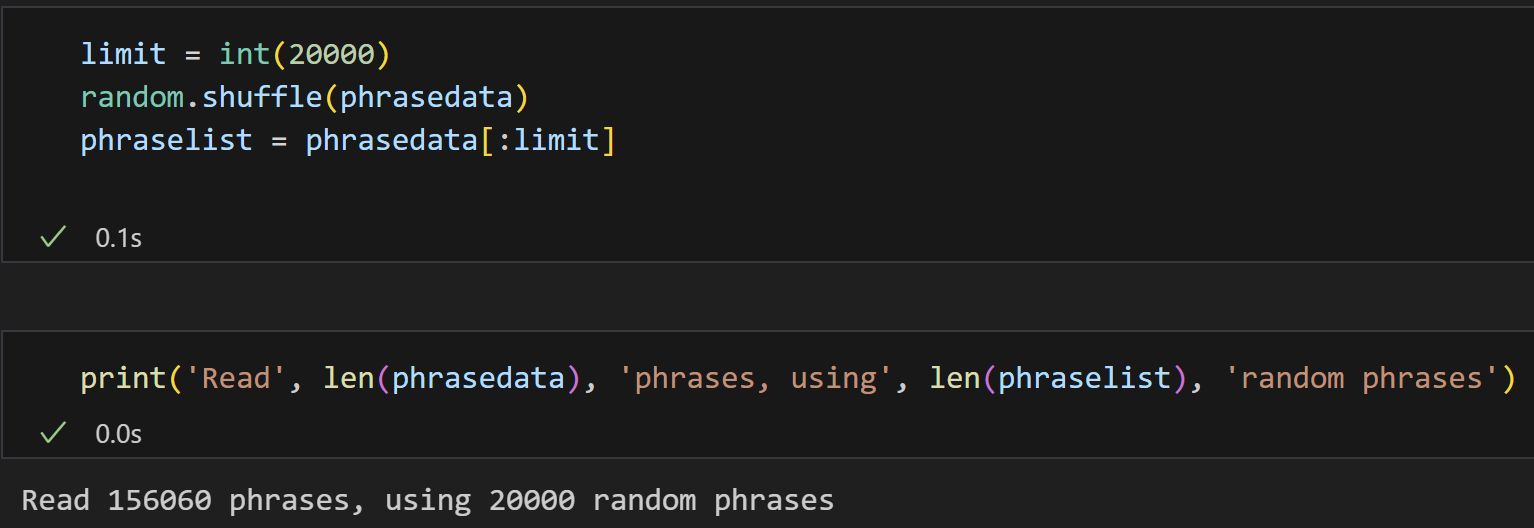


Take a Glimpse of the Dataset

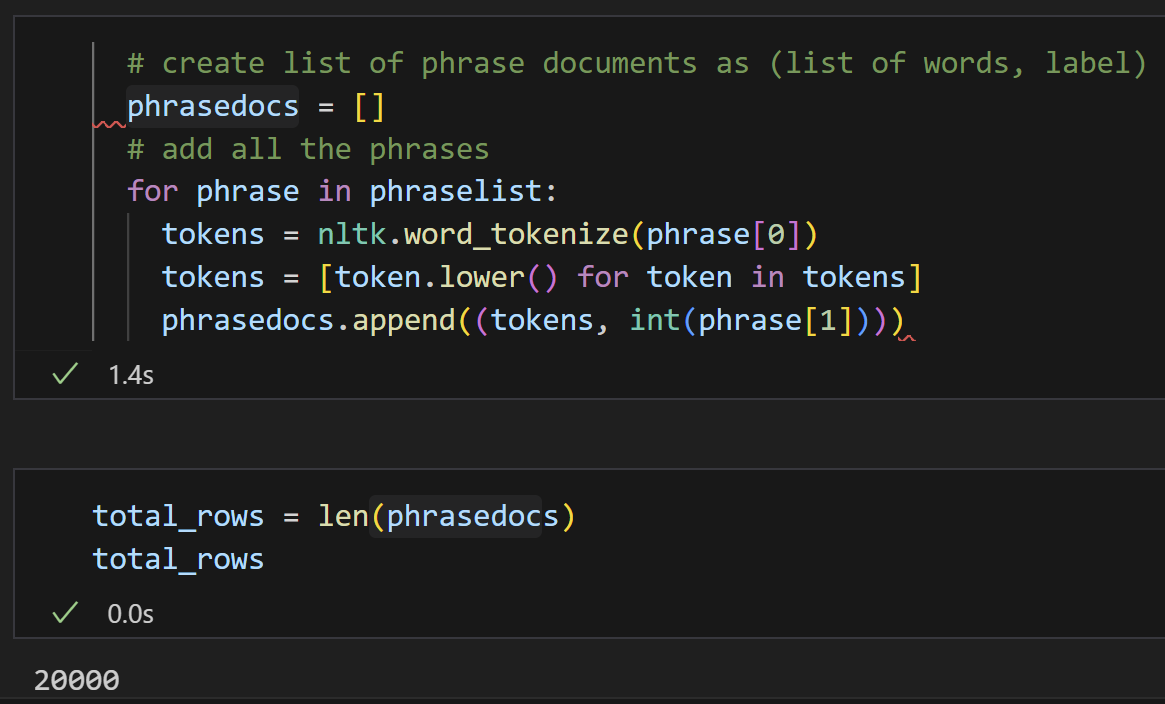


I decided to extract the relevant columns, specifically 'Phrase' and 'Sentiment'. This simplifies the dataset and ensures that only the necessary information is used for further processing.

I set a limit of 20,000 random phrases for analysis because my computer’s RAM is limited to only 16GB. Initially, I’ve tried to analyze larger with 30,000 and 50,000 random phrases, but these attempts consistently resulted in memory errors. To address this issue, I reduced the dataset size to ensure that the analysis could be conducted successfully without exceeding the available memory.

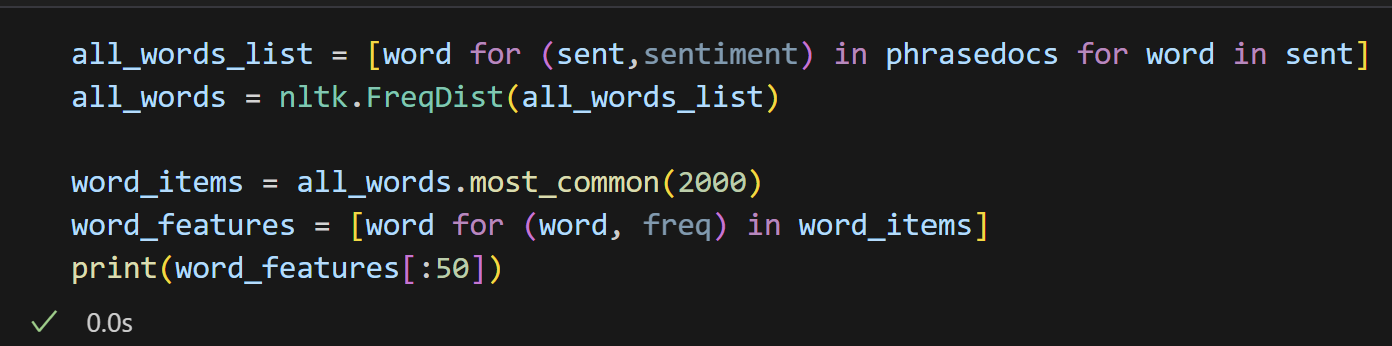


Following the steps in the template, I created a phrasedoc. Each phrase in the dataset was tokenized using the word\_tokenize function. Additionally, I included an extra step to convert all tokens to lowercase, ensuring consistency in the text preprocessing

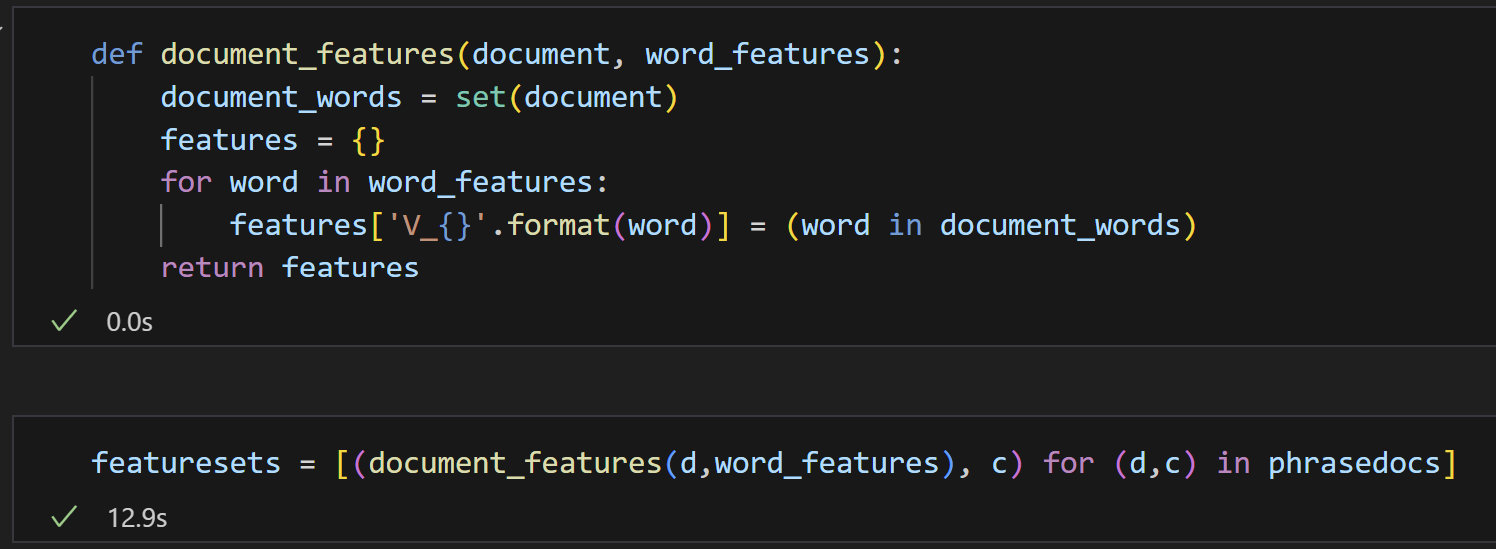


1. **Analysis**
   1. **Without any processing.**

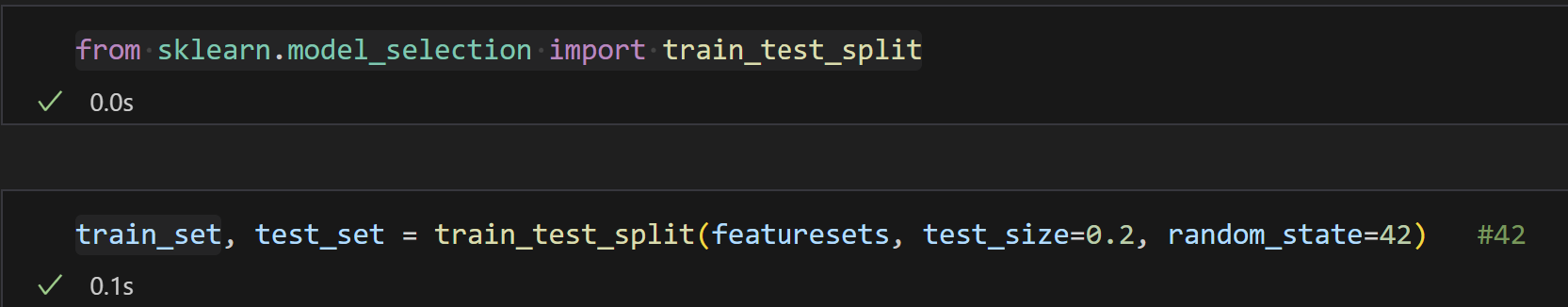
With the dataset and tokens prepared, I proceeded to create the first set of features using the bag-of-words approach. This step serves as a baseline model without any additional processing. I selected the 2,000 most common words from the dataset to form the feature set.



I defined the document\_features function to extract features from each document.

Using this function, I generated featuresets, which pairs the feature dictionary of each document with its corresponding sentiment label. This structured data is then ready for use in training and evaluating the model

I used the train\_test\_split function to divide the feature sets into two parts: 80% for the training set and 20% for the validation set. This ensures that the model is trained on a majority of the data while reserving a portion for evaluating its performance.

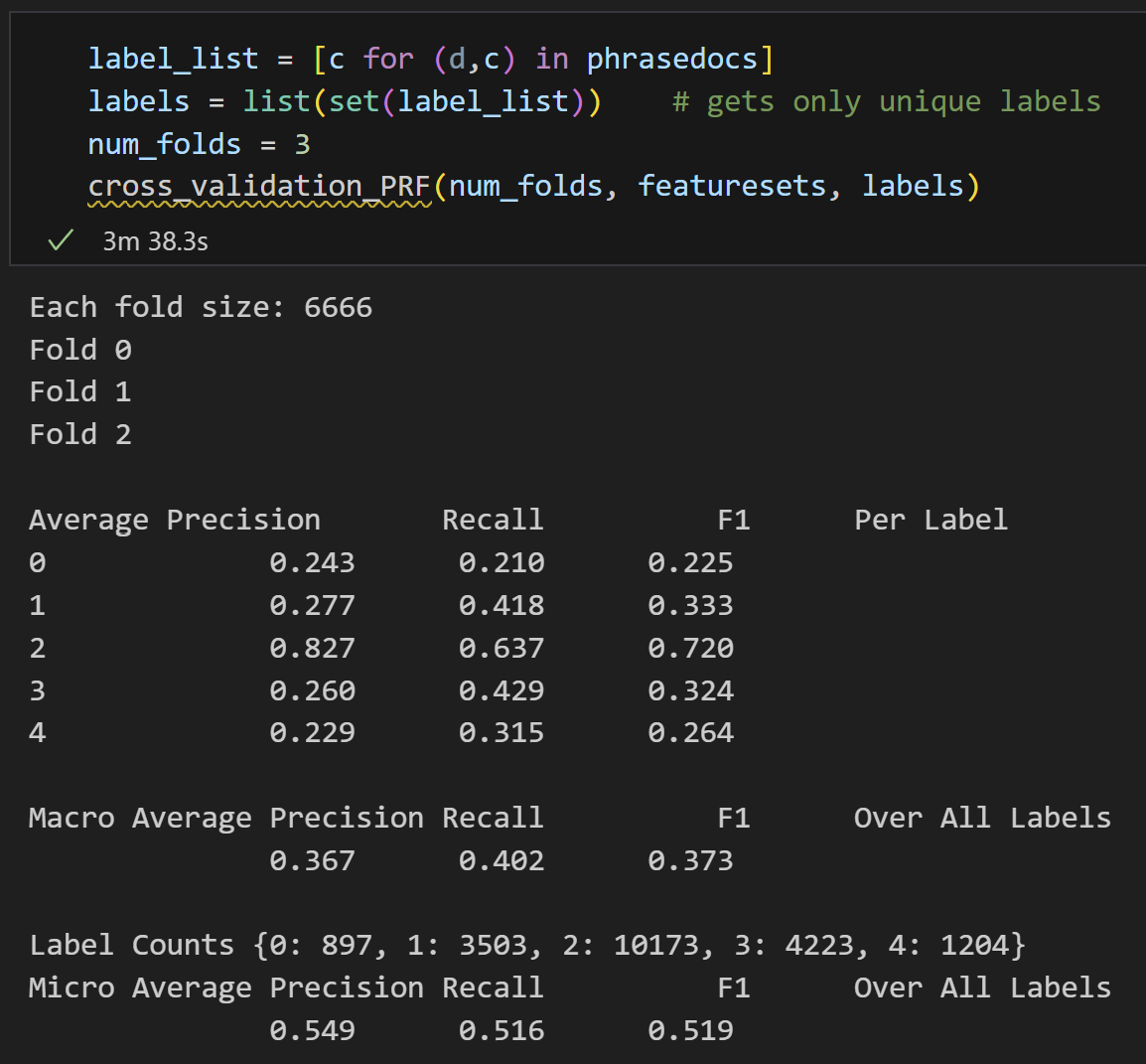


I trained a Naive Bayes Classifier on the training set using the NLTK library. The model achieved an accuracy of 0.56725 (approximately 56.73%) on the validation set. This serves as the baseline performance for the sentiment analysis task, using the bag-of-words approach without additional preprocessing



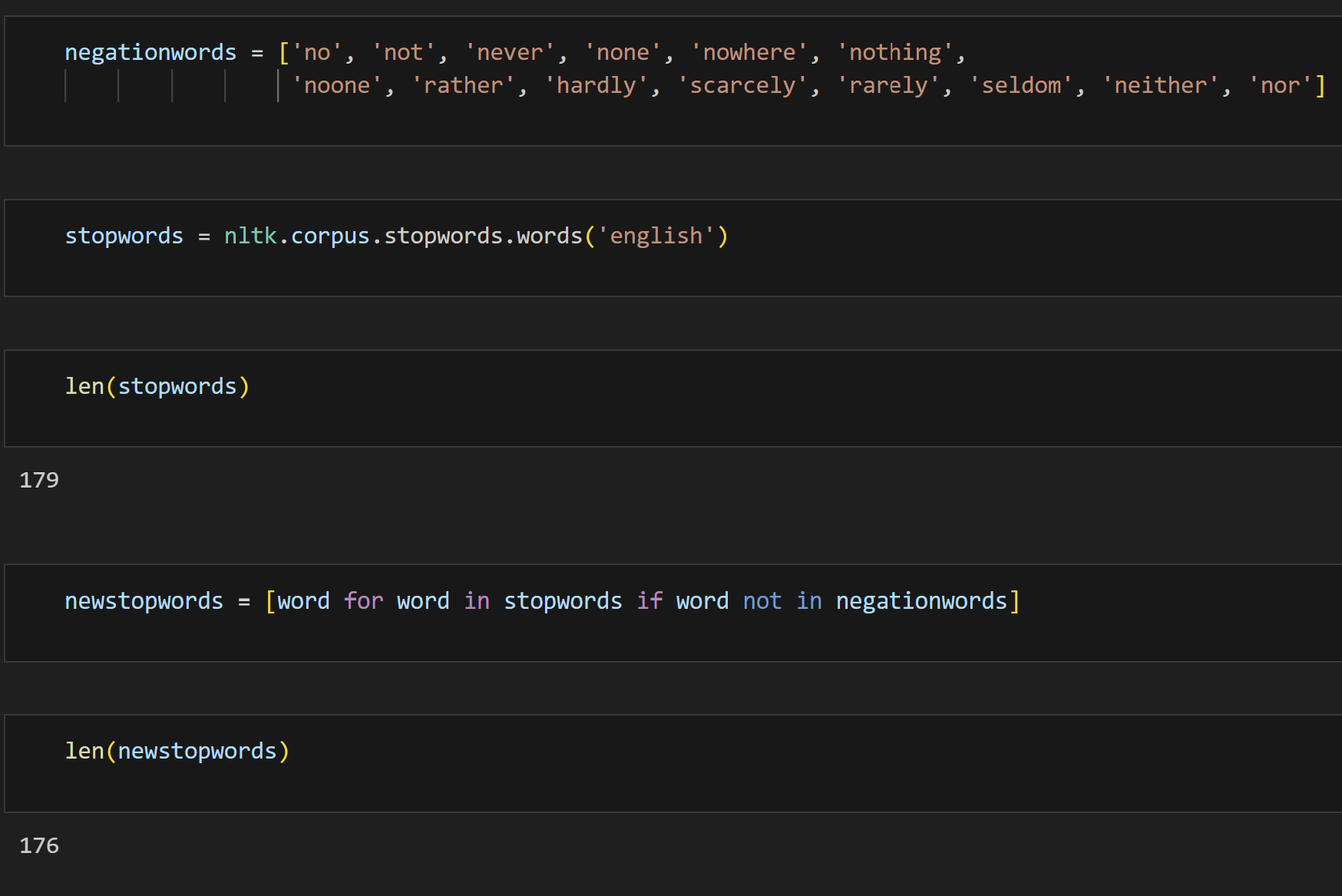
Next, I performed a 3-fold cross-validation on the dataset to evaluate the model's performance across different splits. The dataset was divided into three equal parts, each containing 6,666 samples. Each fold alternated between serving as the validation set and being part of the training set.

The model performed best for label 2, with an F1-score of 0.720, likely due to its higher representation in the dataset. Labels 0, 4, and 1 showed relatively lower performance, possibly because of their smaller sample sizes.

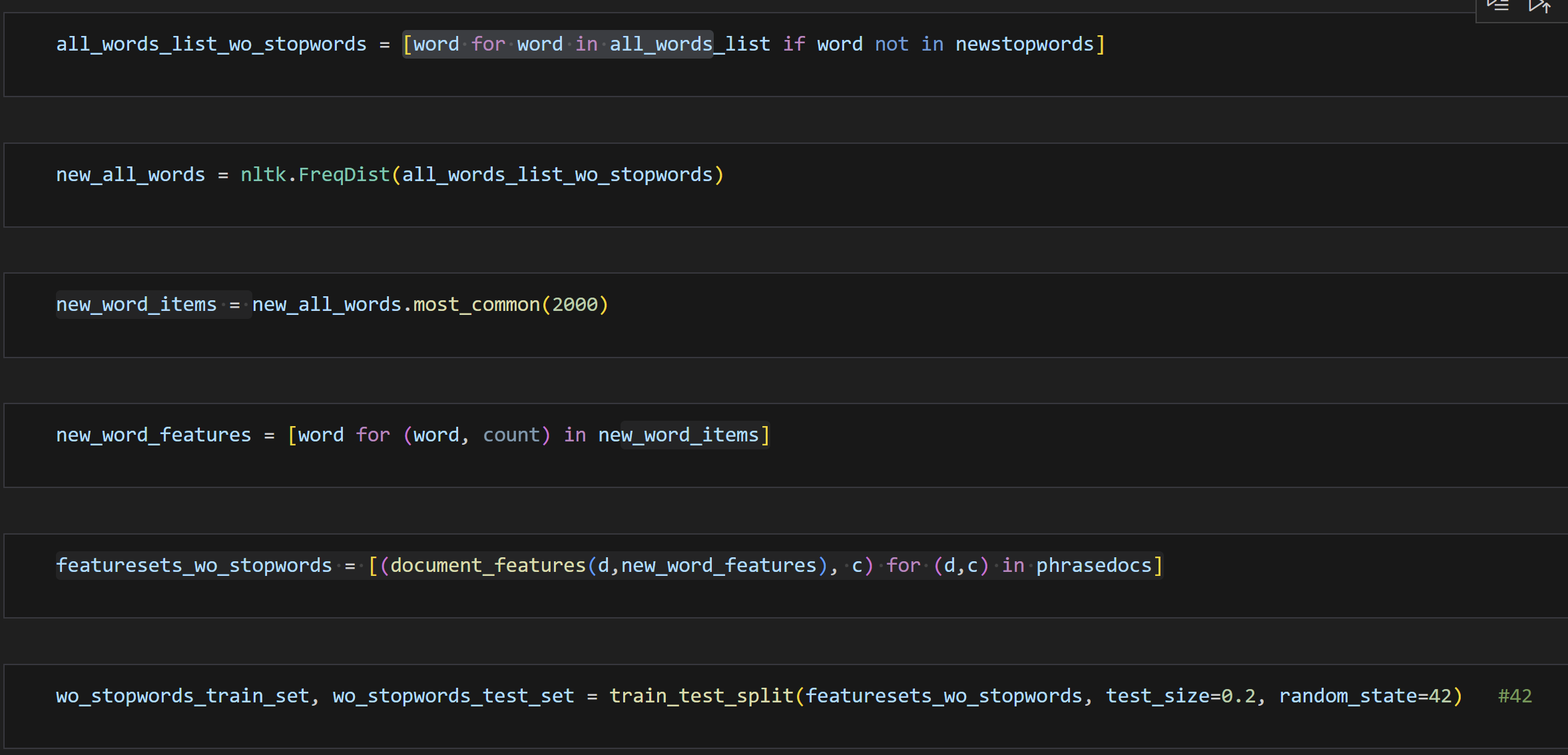


* 1. **Remove stop words**

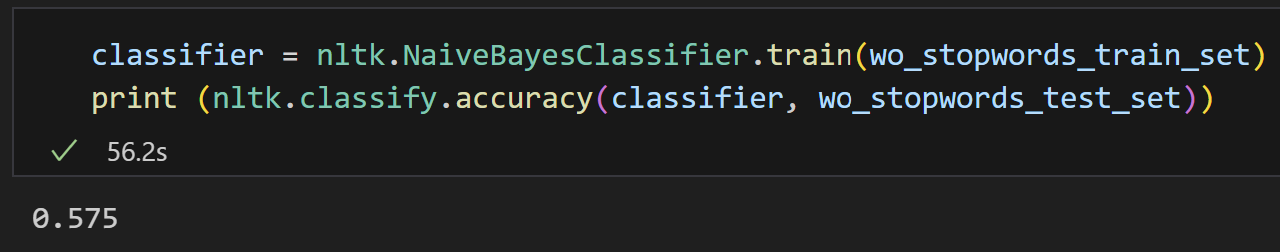
Before removing all the stop words, I chose to exclude negation words from the stop words list. Because I plan to handle negation words separately later in the process. By doing this, I ensure that the removal of stop words does not interfere with the negation-related features, which are critical for sentiment analysis.



After removing the stop words from the all words list, I recreated the Bag-of-Words features using the 2,000 most frequent words from the filtered dataset. This step aims to evaluate whether excluding stop words improves the accuracy of the model compared to the baseline performance.

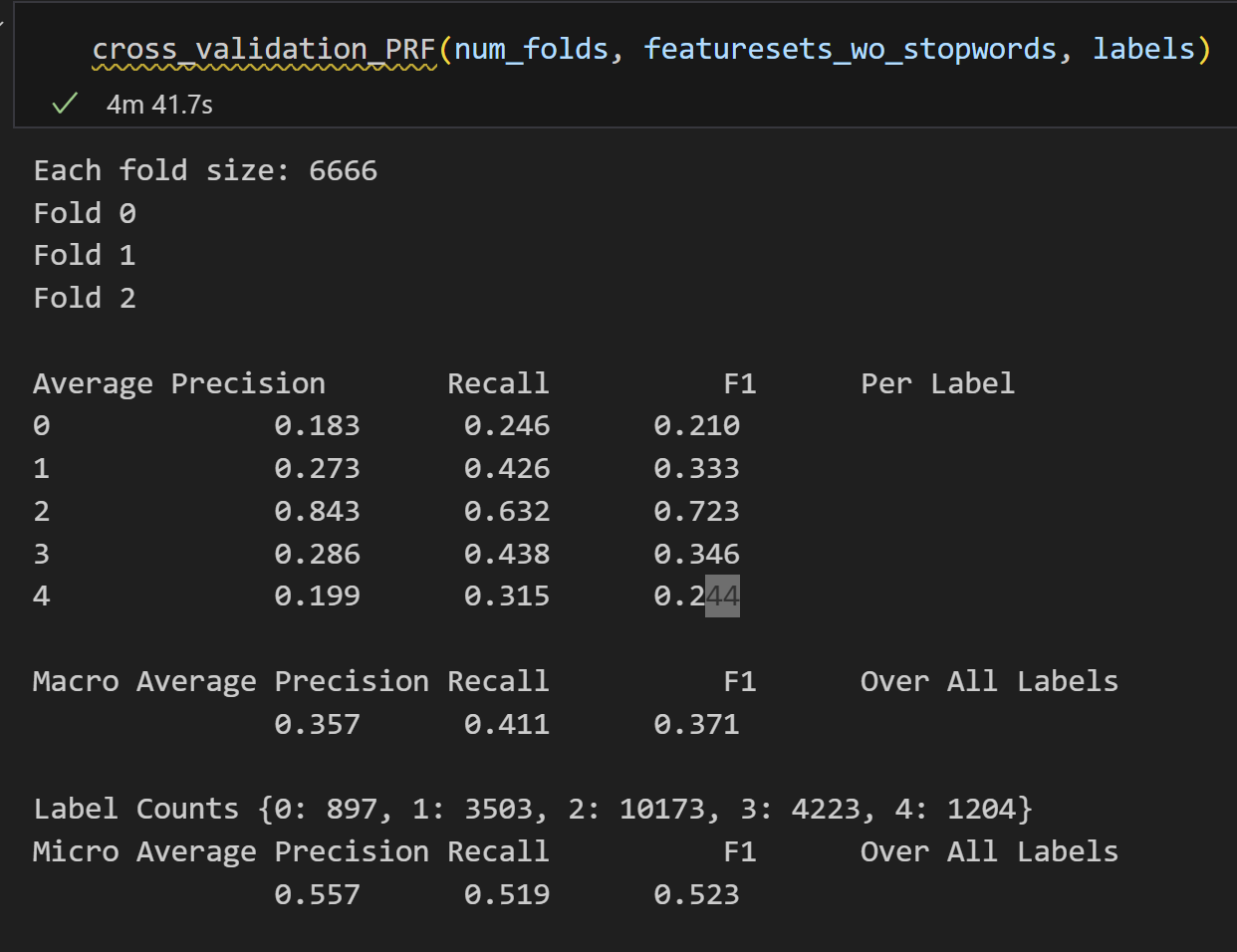


After training the Naive Bayes Classifier on the dataset without stop words, the model achieved a slightly improved accuracy of 0.575 (57.5%) on the test set. This indicates that removing stop words helped enhance the model's performance. This improvement suggests that excluding irrelevant tokens can make the features more meaningful for sentiment classification.

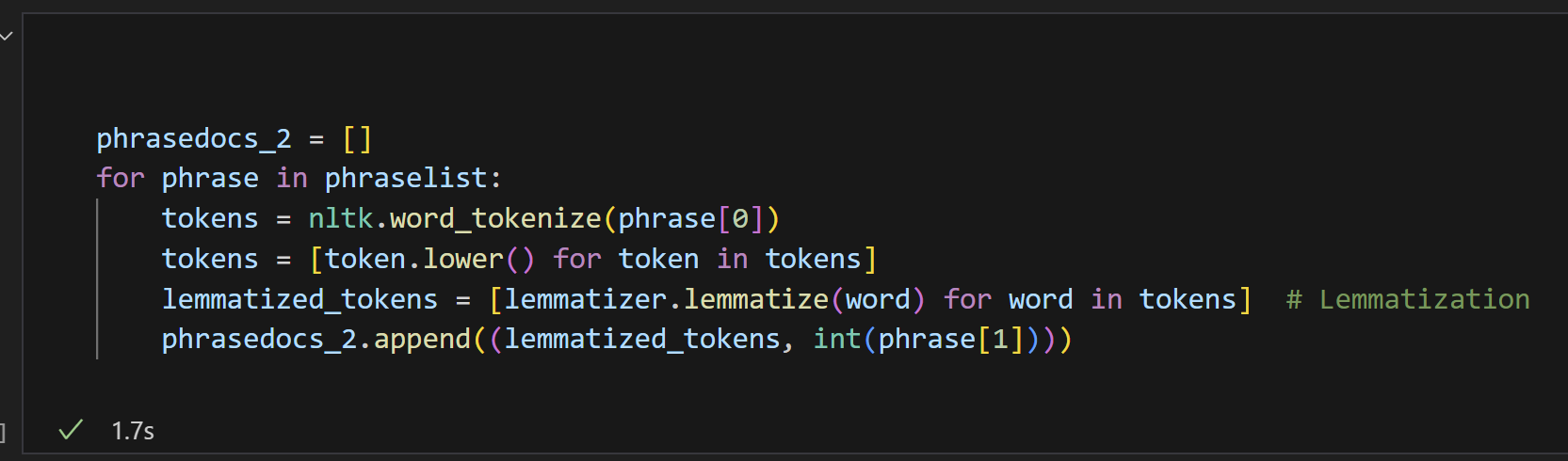


Let’s take a look at cross validation:

Compared to the results with stopwords, the overall performance shows a slight improvement, particularly in the Micro Average metrics. This suggests that removing stop words reduces noise.



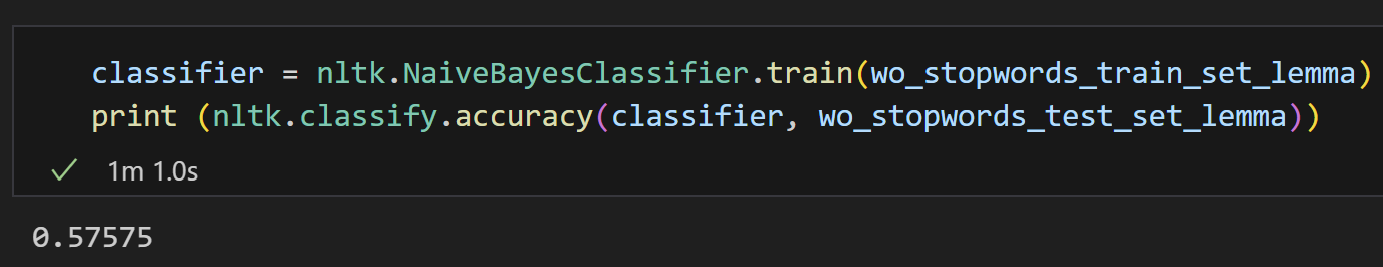
* 1. **Lemmatization → Removing Stopwords → Bag-of-Words**I created a new phrasedocs\_2 by applying lemmatization to the tokenized text. This process involves converting each word into its base or dictionary form (e.g., 'running' → 'run') using the WordNet Lemmatizer. By doing this, I aim to normalize the text and reduce redundant variations of words.



Using the lemmatized tokens from the updated phrasedocs\_2, I constructed a new feature set featuresets\_wo\_stopwords\_lemma by applying the Bag-of-Words approach with the most frequent words, excluding stop words. This feature set was then split into training and testing sets (80% training, 20% testing) using the train\_test\_split function with a fixed random seed, 42, to ensure reproducibility.

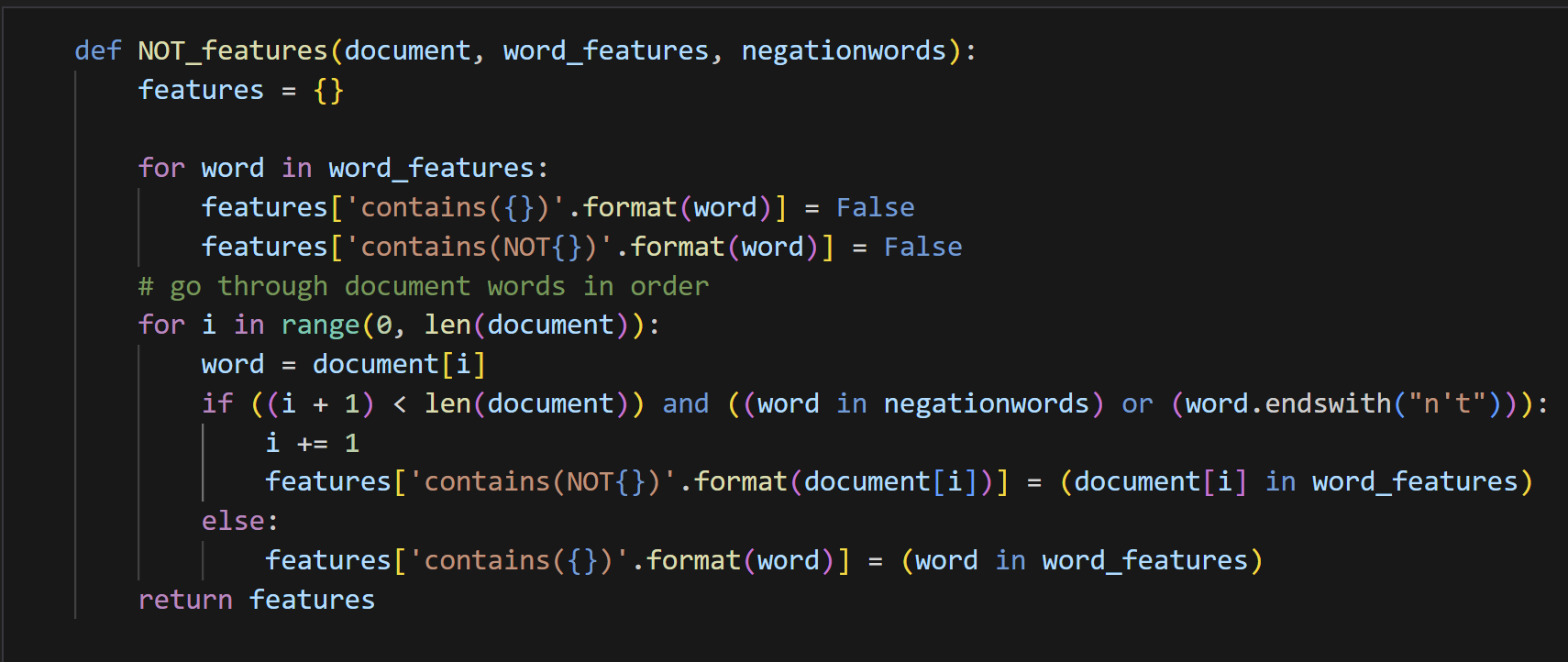


After training the Naive Bayes Classifier on the lemmatized feature set (phrasedocs\_2), the model achieved a slightly improved accuracy of 0.57575 (57.58%) on the test set. Although the improvement is small, it indicates that lemmatization contributes positively to the model's performance. Therefore, I decided to proceed with phrasedocs\_2 for further analysis while retaining both lemmatization and stop word removal as part of the preprocessing pipeline.

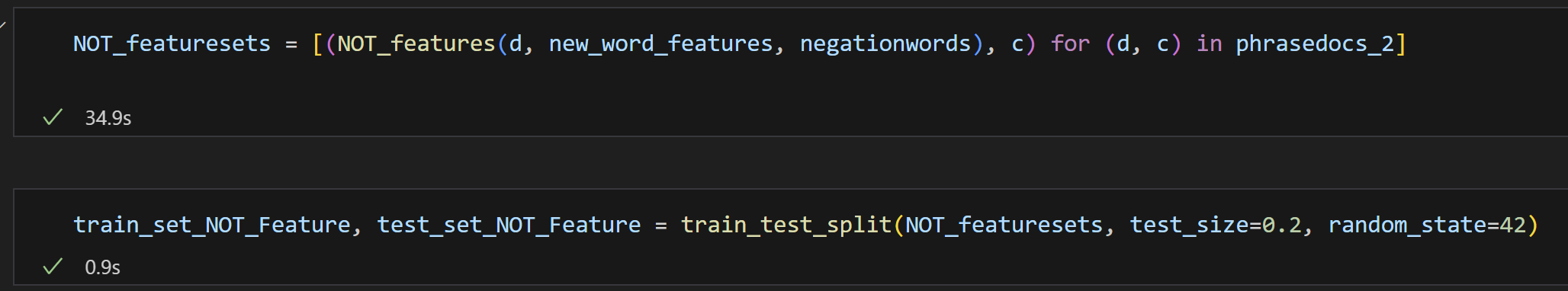


* 1. **Using negation features.**

I defined the NOT\_features function to account for negation in the dataset. This function is designed to enhance Bag-of-Words feature extraction by identifying and handling negation words, ensuring that the model captures the sentiment-altering effect of negation in text.

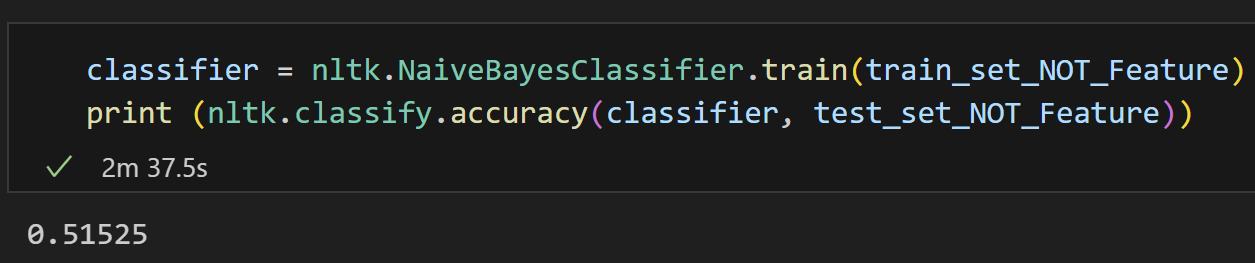


I created a new feature set, NOT\_featuresets, by applying the NOT\_features function to the lemmatized dataset (phrasedocs\_2). This step incorporates negation handling into the feature extraction process by capturing both regular word features and negation-affected features. The dataset was then split into training and testing sets (train\_set\_NOT\_Feature and test\_set\_NOT\_Feature) using an 80-20 split with a fixed random seed (42) for reproducibility.



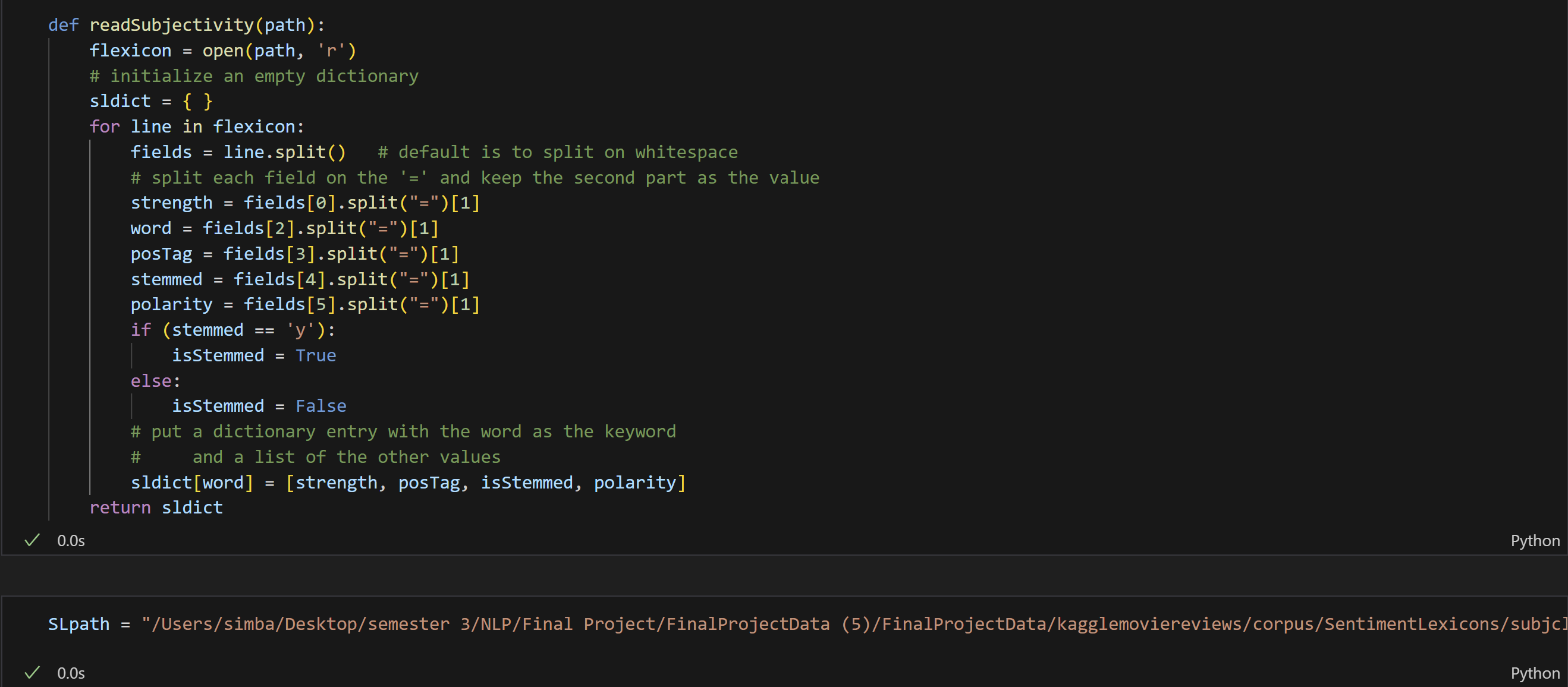
After training the Naive Bayes Classifier using the feature set with incorporated negation handling (NOT\_featuresets), the model achieved an accuracy of 0.51525 (51.53%) on the test set. Unfortunately, this represents a decrease in performance compared to the previous approach without explicit negation features.

This result suggests that while handling negation is conceptually important, the current implementation of negation handling might have introduced additional complexity or noise, negatively impacting the classifier's performance.

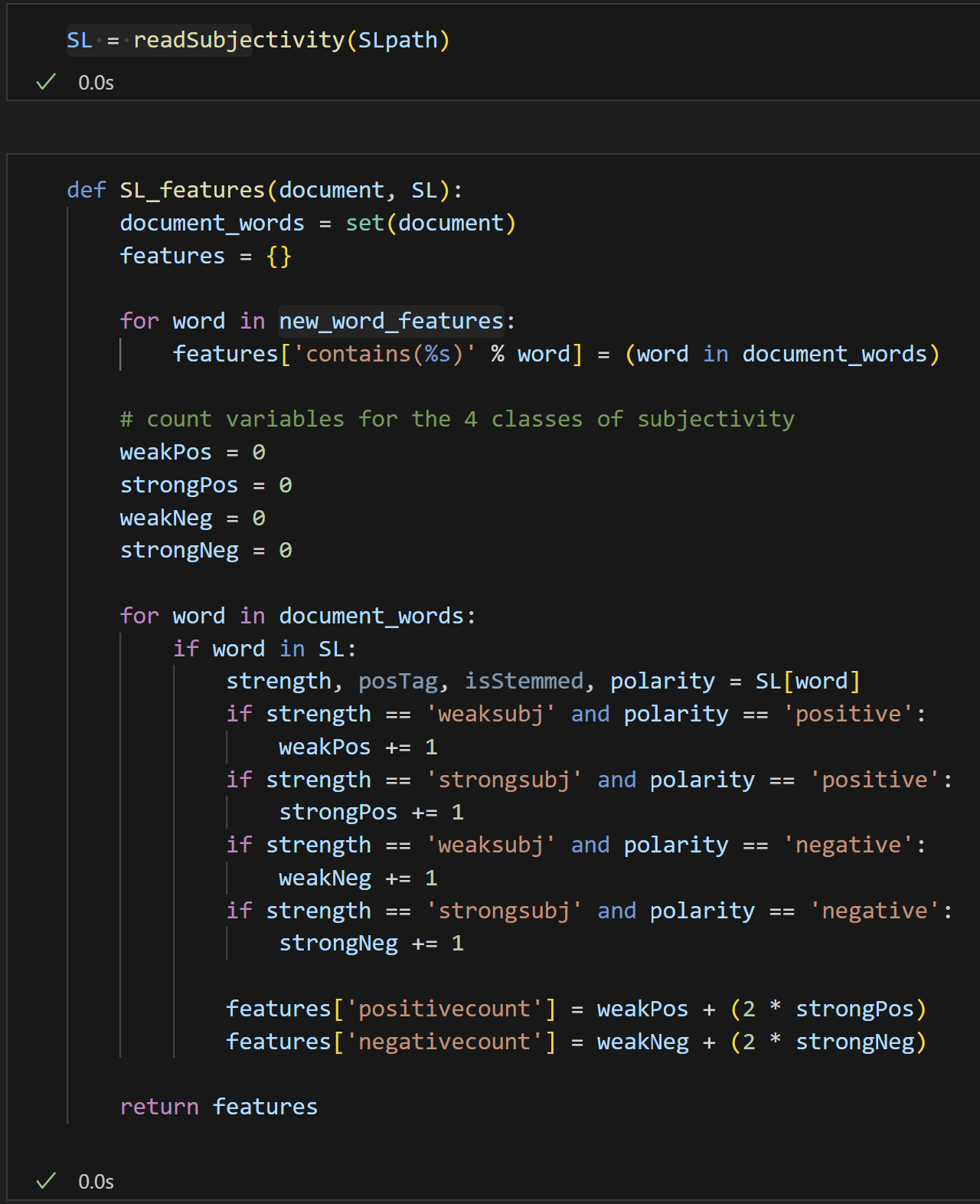


* 1. **Subjectively: SL**

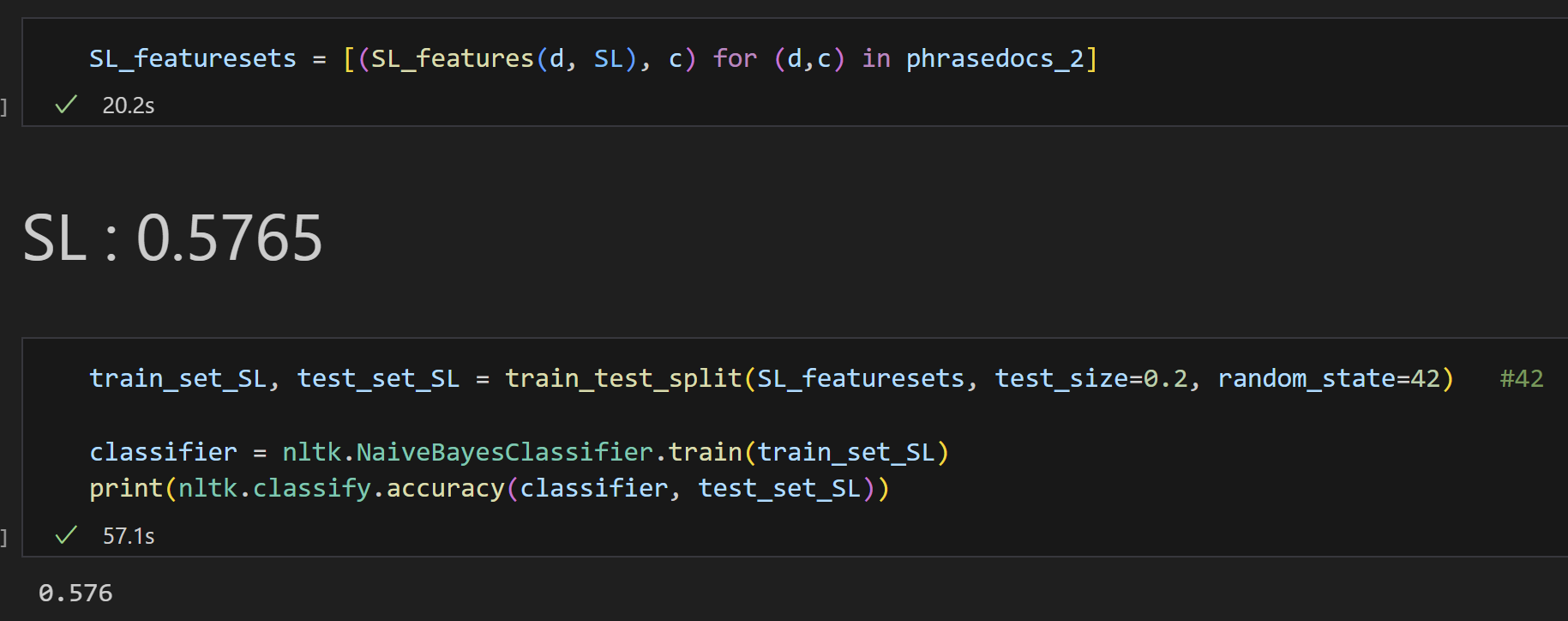
Set up the readSubjectivity and setup the SL Path the read the file



After we read the file, I define the SL\_features.

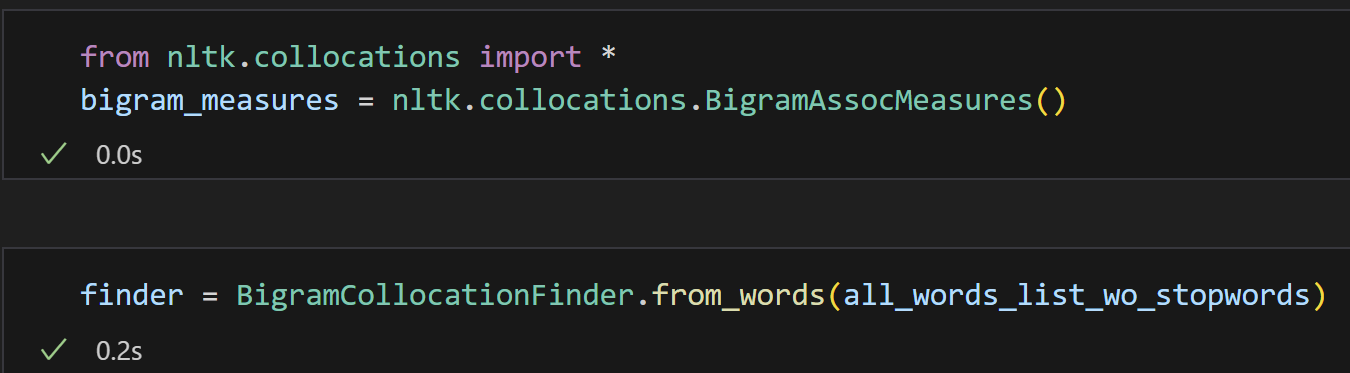


After integrating subjectivity features from the Subjectivity Lexicon (SL) into the feature set, I trained the Naive Bayes Classifier and achieved an accuracy of 0.576 (57.6%) on the test set. This represents a slight improvement compared to the baseline accuracy, indicating that subjectivity features contribute positively to the model's ability to analyze sentiment.

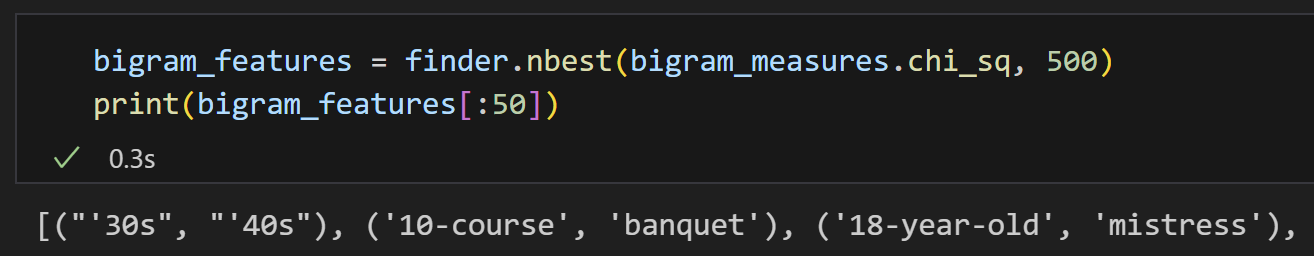


* 1. **Bi-grams into Feature Extraction**

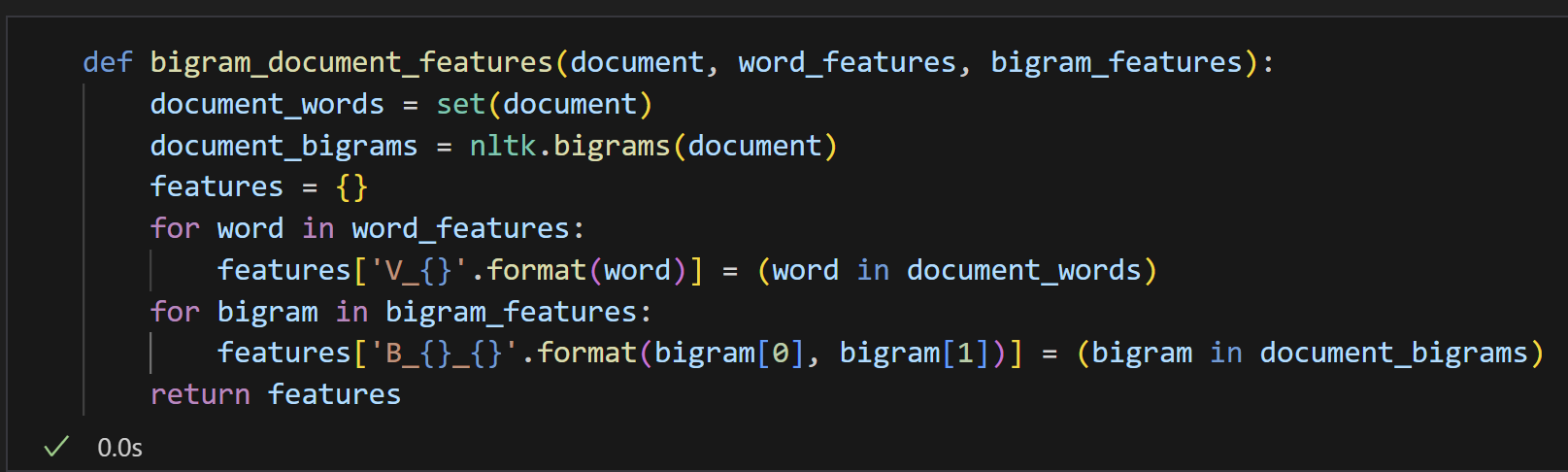
To capture contextual relationships between consecutive words, I incorporated bi-grams into the feature extraction process. Using the BigramCollocationFinder from NLTK, I identified the most significant bi-grams based on their chi-squared scores.   
Extracted bi-grams from the preprocessed word list (all\_words\_list\_wo\_stopwords), which excludes stop words. Used the BigramCollocationFinder to analyze word pair combinations.



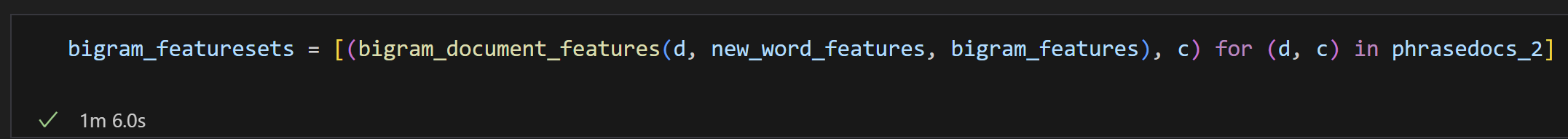
Ranked bi-grams by their chi-squared scores using finder.nbest. Selected the top 500 bi-grams as features for further analysis.



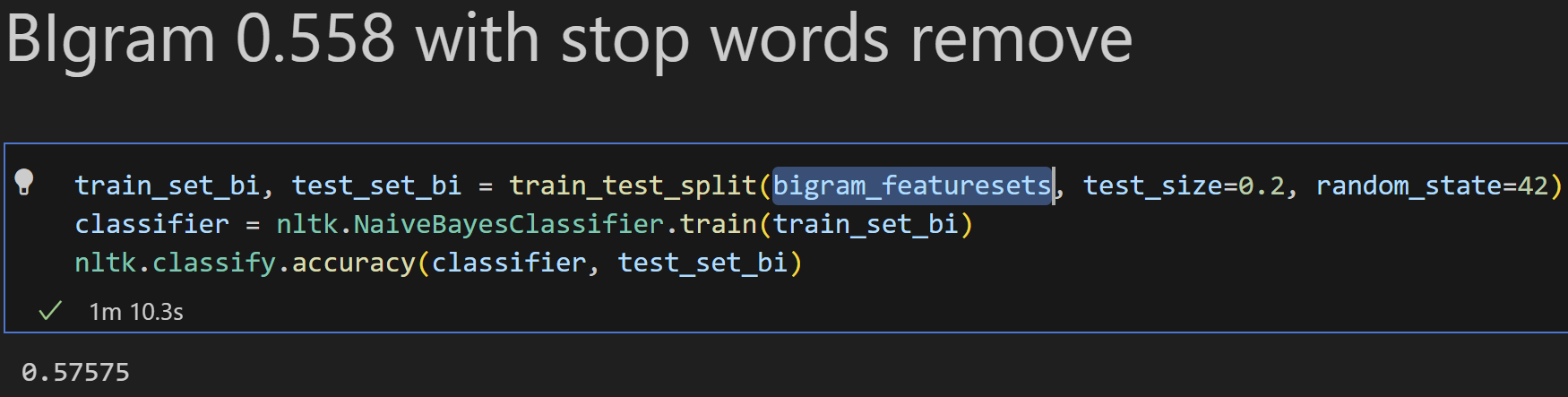
I created the bigram\_document\_features function to combine uni-gram and bi-gram features in the feature extraction process. This function enhances the Bag-of-Words approach by incorporating bi-grams, which capture contextual relationships between consecutive words.



I created bigram\_featuresets by applying the bigram\_document\_features function to the lemmatized dataset (phrasedocs\_2). This feature set combines uni-grams (individual words) and bi-grams (word pairs) to capture both isolated word occurrences and contextual relationships within the text.

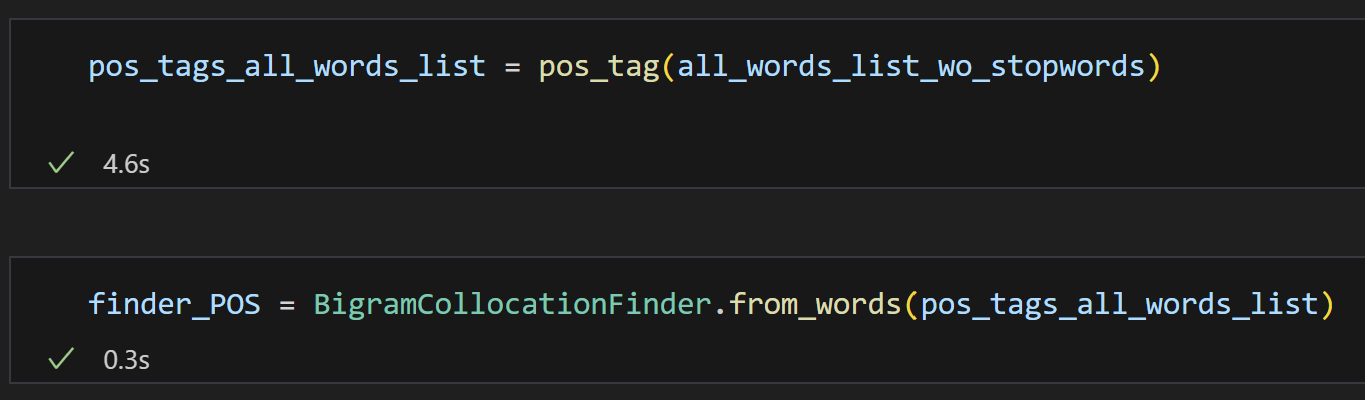


After training the Naive Bayes Classifier on the bi-gram feature set, the model achieved an accuracy of 0.57575 (57.58%) on the test set. This result indicates that incorporating bi-grams did not significantly improve performance compared to uni-grams alone. I am uncertain why the accuracy did not improve as expected. Further experimentation, combining bi-grams with other features, may help achieve better results.

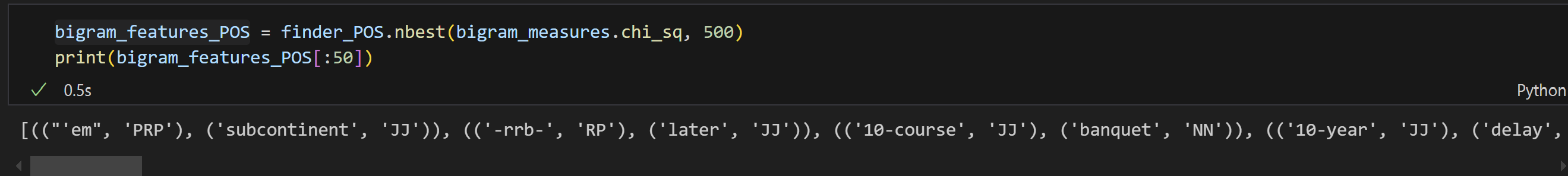


* 1. **POS Tags Followed by Bi-grams**

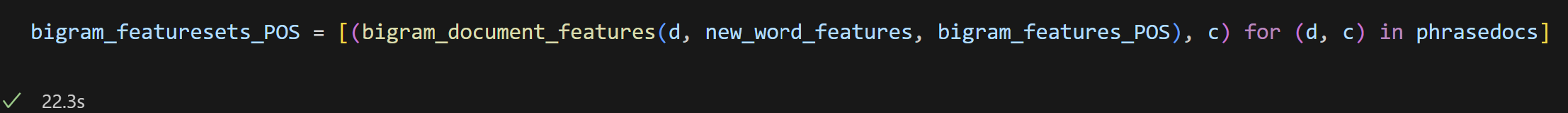
In this step, I applied POS tagging to the preprocessed dataset (all\_words\_list\_wo\_stopwords) to capture the grammatical roles of words. Using NLTK's pos\_tag function, I generated a list of tuples where each word is paired with its respective Part-of-Speech (POS) tag.

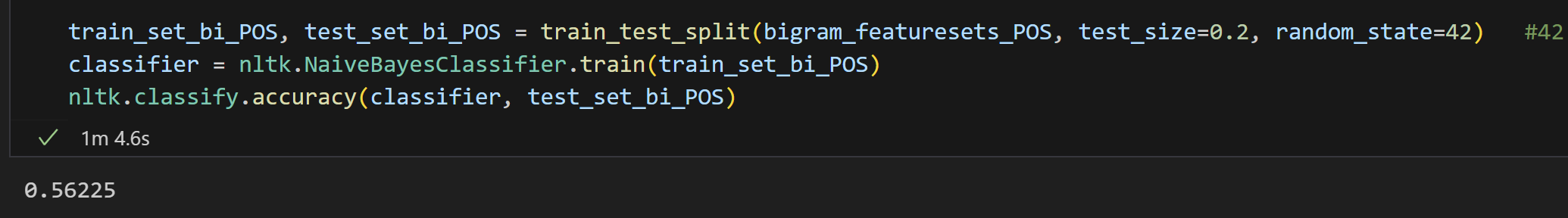


Using the bi-grams generated from the POS-tagged word list, I applied a chi-squared test to identify the top 500 most significant bi-grams.



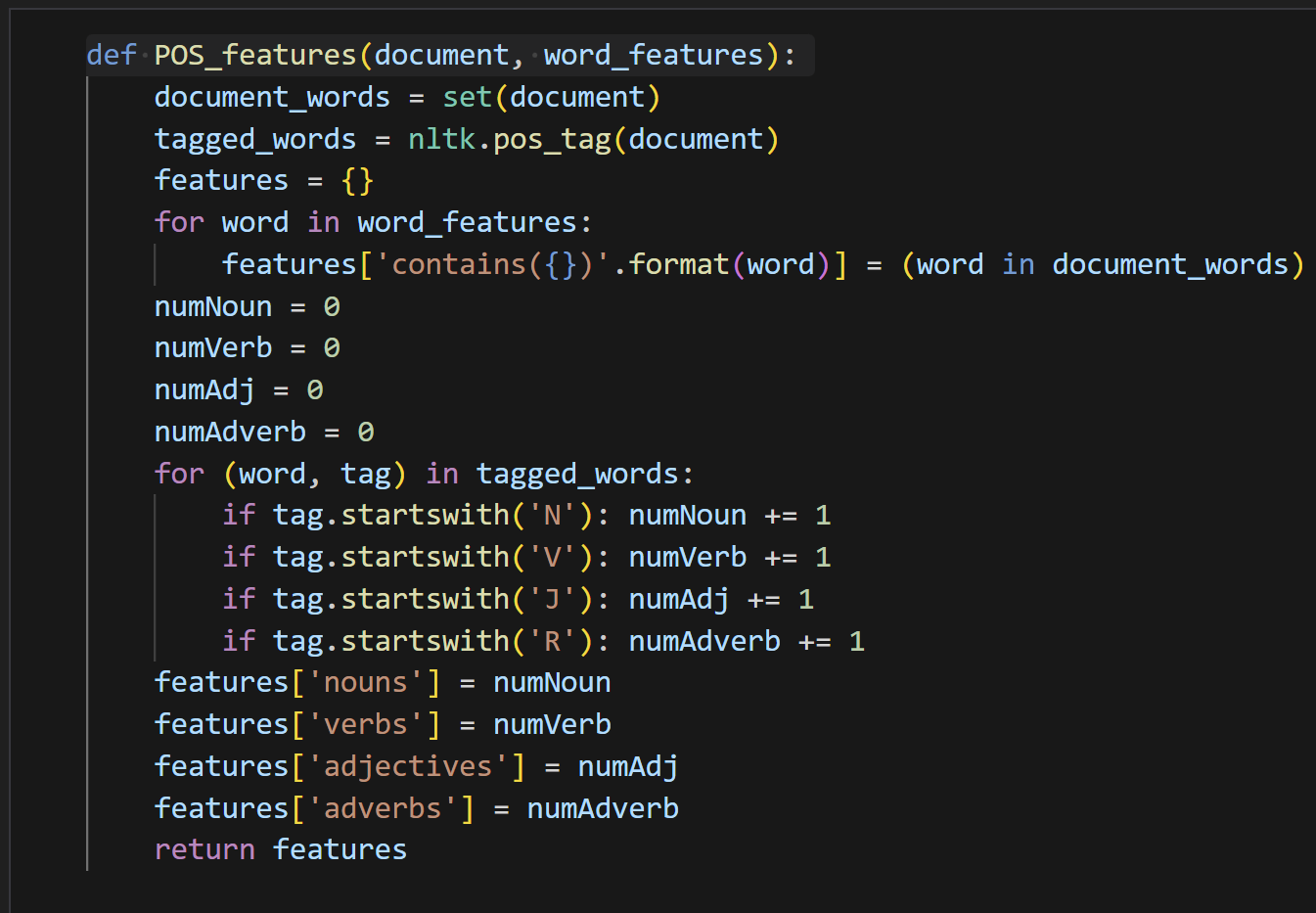
To better align with POS tagging accuracy, I modified the pipeline to use the original dataset without applying lemmatization. I generated the bigram\_featuresets\_POS by combining uni-gram features (new\_word\_features) and the most significant POS-tagged bi-grams (bigram\_features\_POS) extracted from the original tokenized dataset.



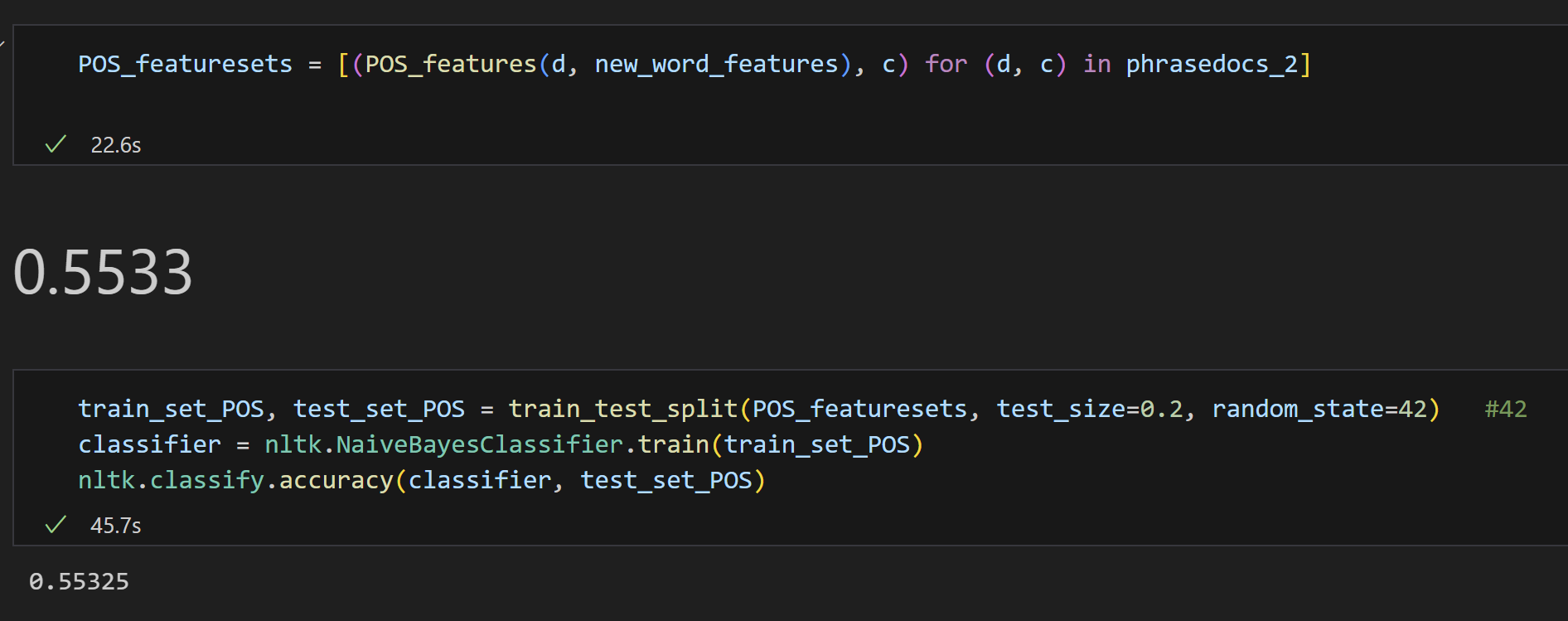
After training the Naive Bayes Classifier on the bigram\_featuresets\_POS generated from the original dataset (without lemmatization), the model achieved an accuracy of 0.56225 (56.23%) on the test set. This result is slightly lower than the accuracy obtained with the bi-gram features using lemmatized data.

* 1. **POS Tag Features**

Set up the function of POS TAG feature

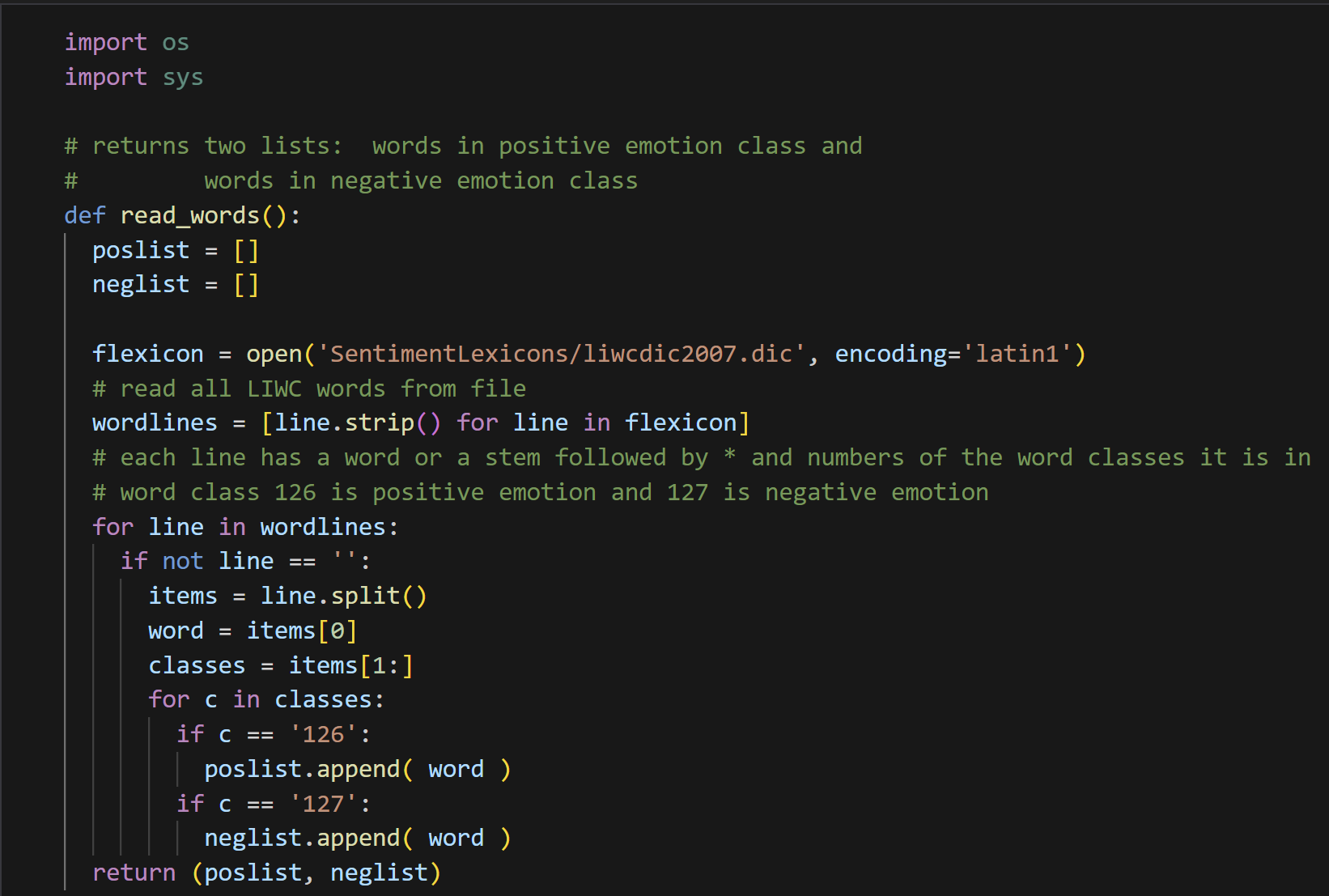


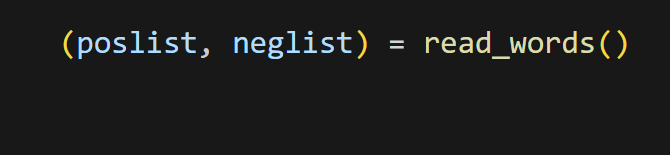
After training the Naive Bayes Classifier on the POS\_featuresets, which were generated using Part-of-Speech (POS) tag features from the original dataset, the model achieved an accuracy of 0.55325 (55.33%) on the test set. This result indicates that incorporating POS tags alone does not significantly improve the model's performance compared to the baseline.

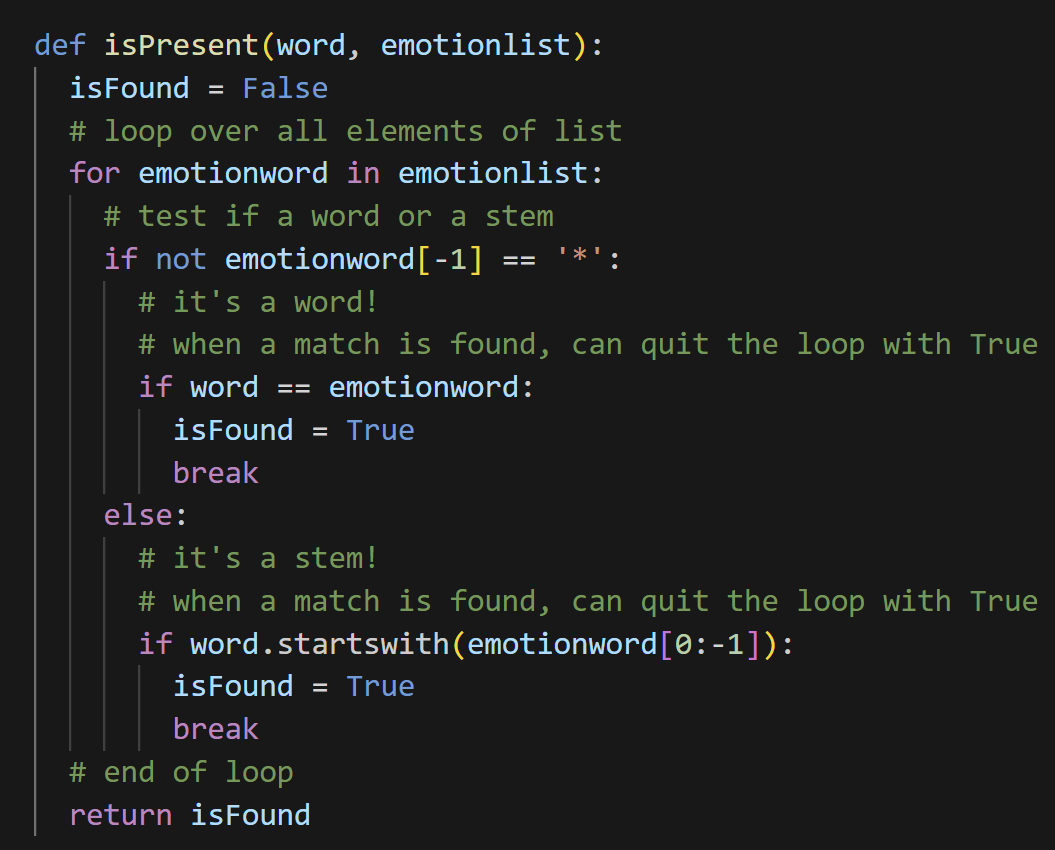


* 1. **LIWC**

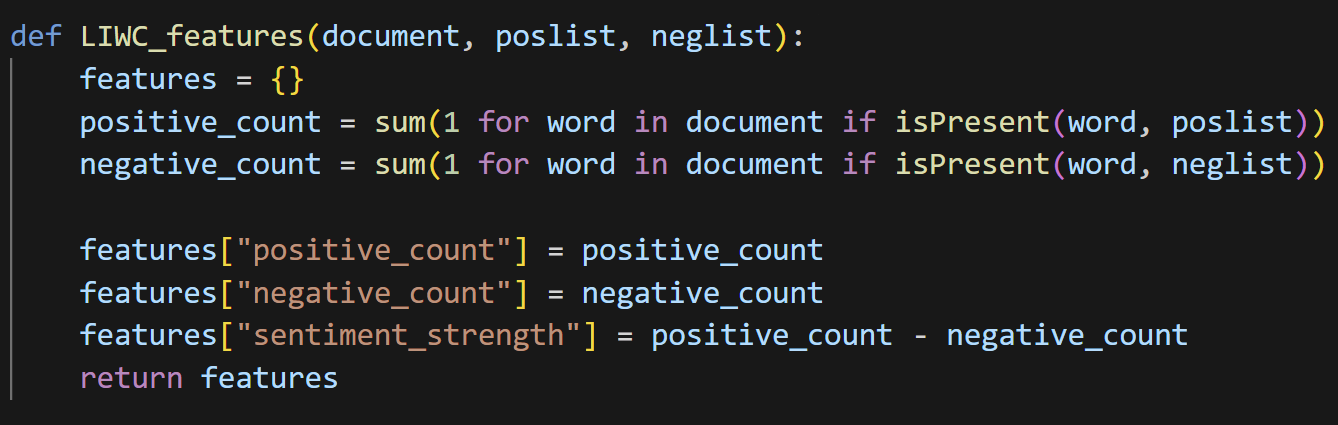
Using the function that already in the file.





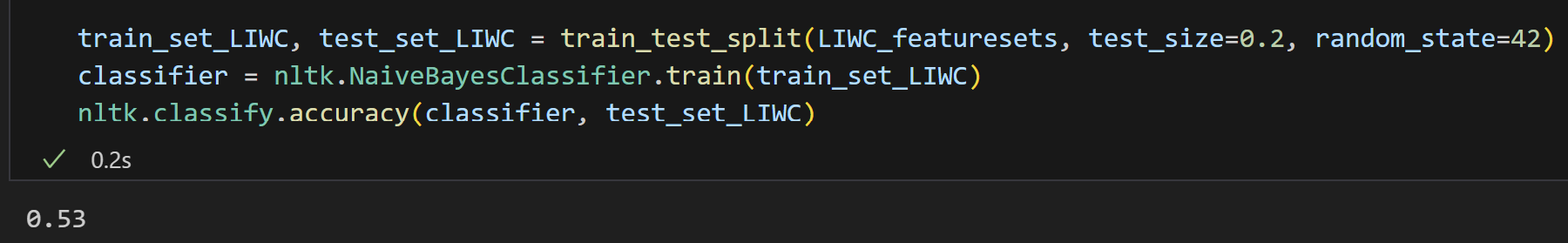


I implemented the LIWC\_features function to generate sentiment-based features using the positive and negative word lists (poslist and neglist) derived from the LIWC sentiment lexicon.

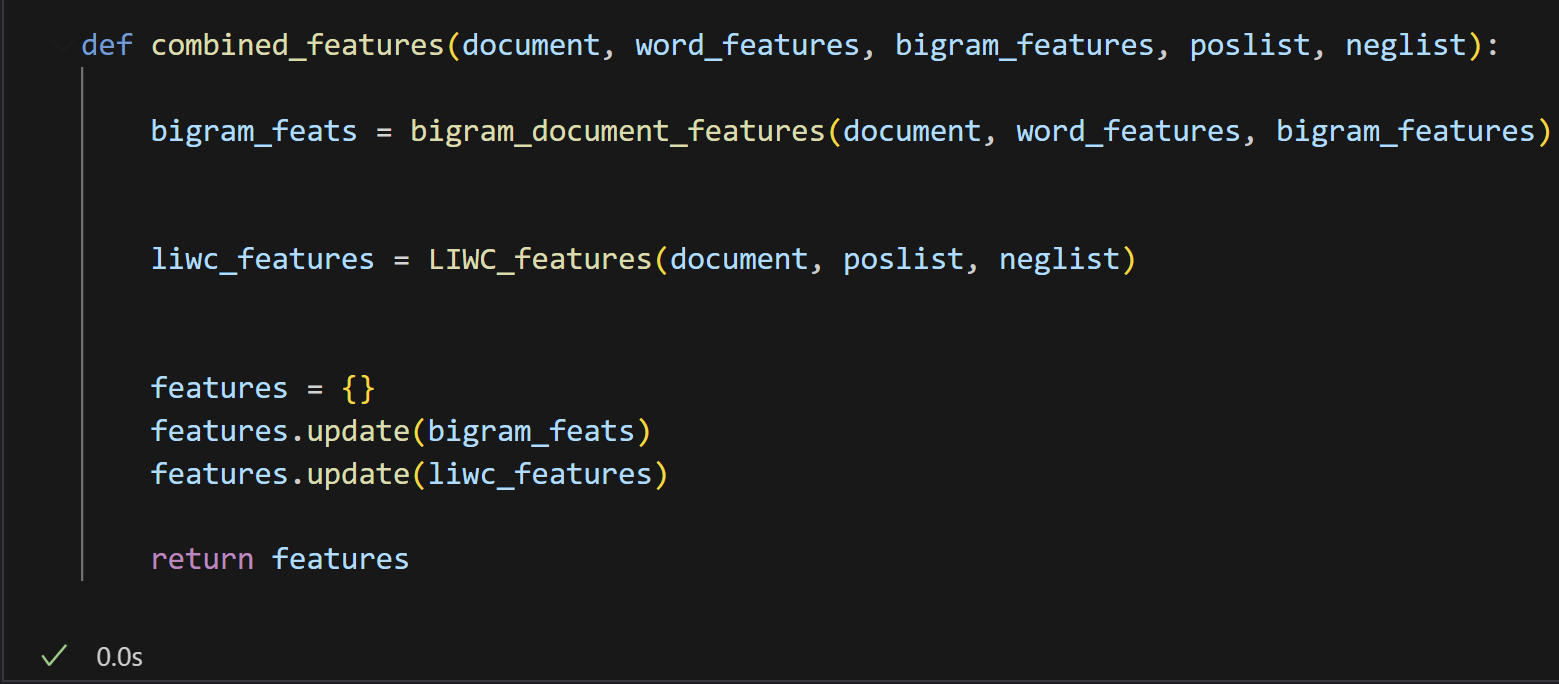


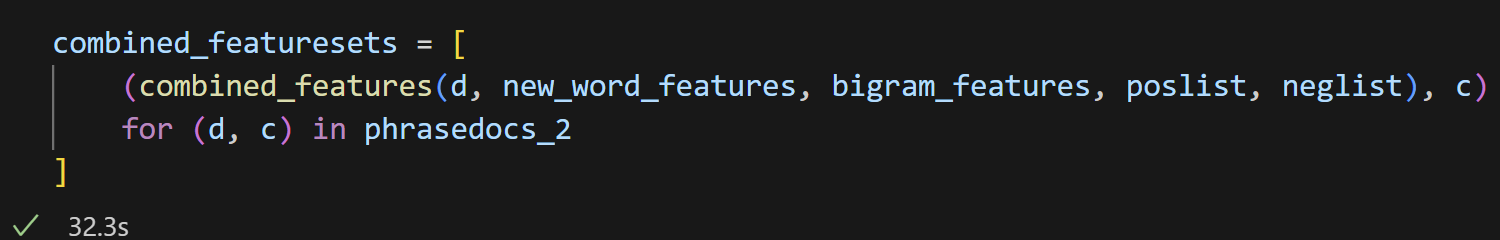
After training the Naive Bayes Classifier using the LIWC-based sentiment features (LIWC\_featuresets), the model achieved an accuracy of 0.53 (53%) on the test set.

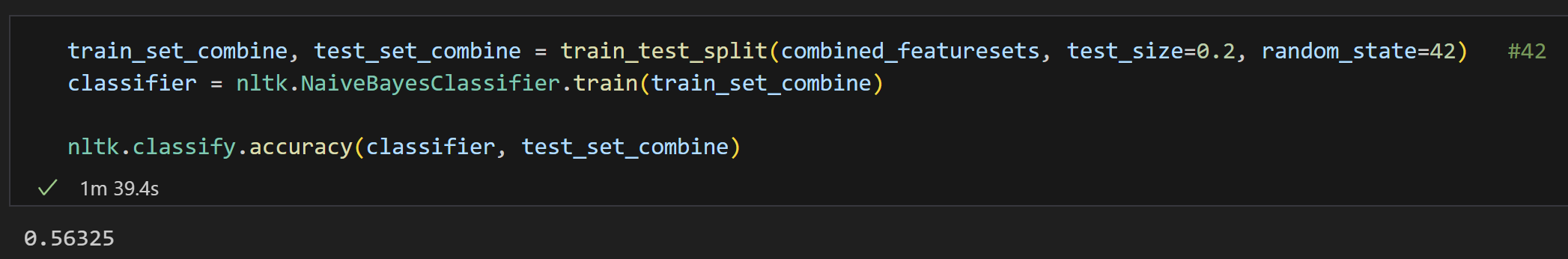
The relatively low accuracy suggests that these features alone may not capture the full complexity of sentiment in the dataset. Combining LIWC features with other feature types, such as uni-grams, bi-grams, or POS-tagged features, could potentially improve performance by adding more contextual and syntactic information by adding more contextual and syntactic information



* 1. **Combine bigram and LIWC**

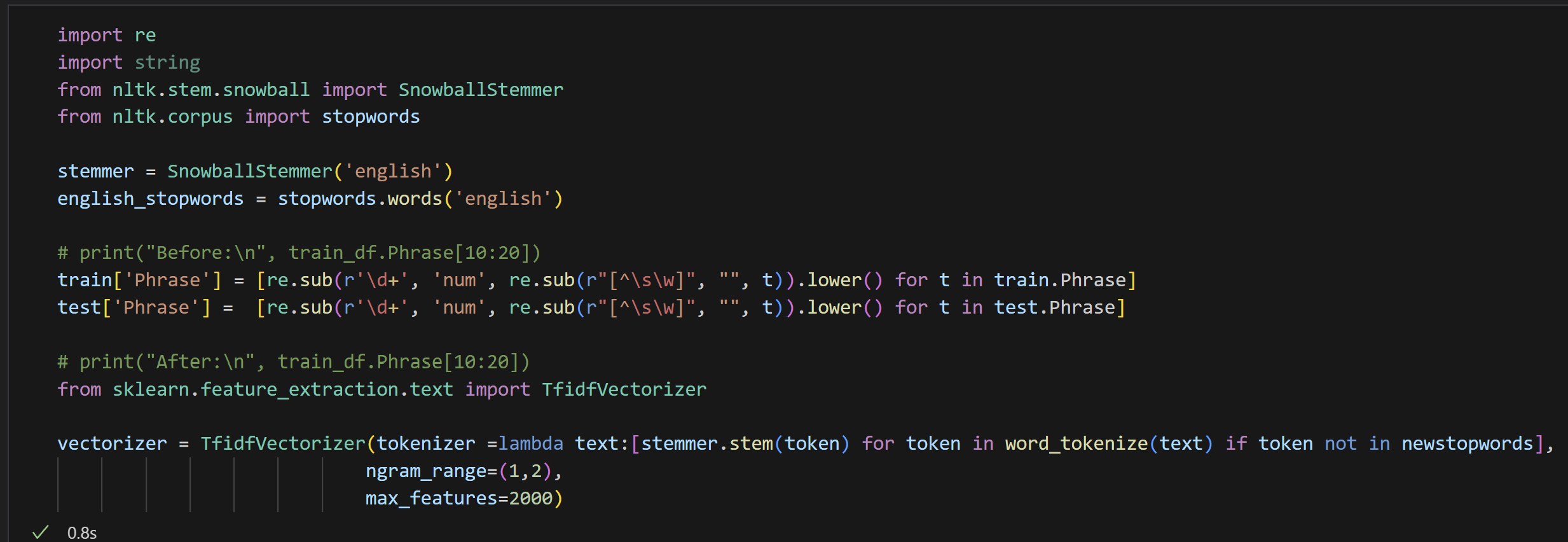
I created the combined\_features function to integrate bi-gram features and LIWC-based sentiment features into a unified feature set. This approach aims to leverage the strengths of both contextual and lexicon-based features for sentiment analysis.

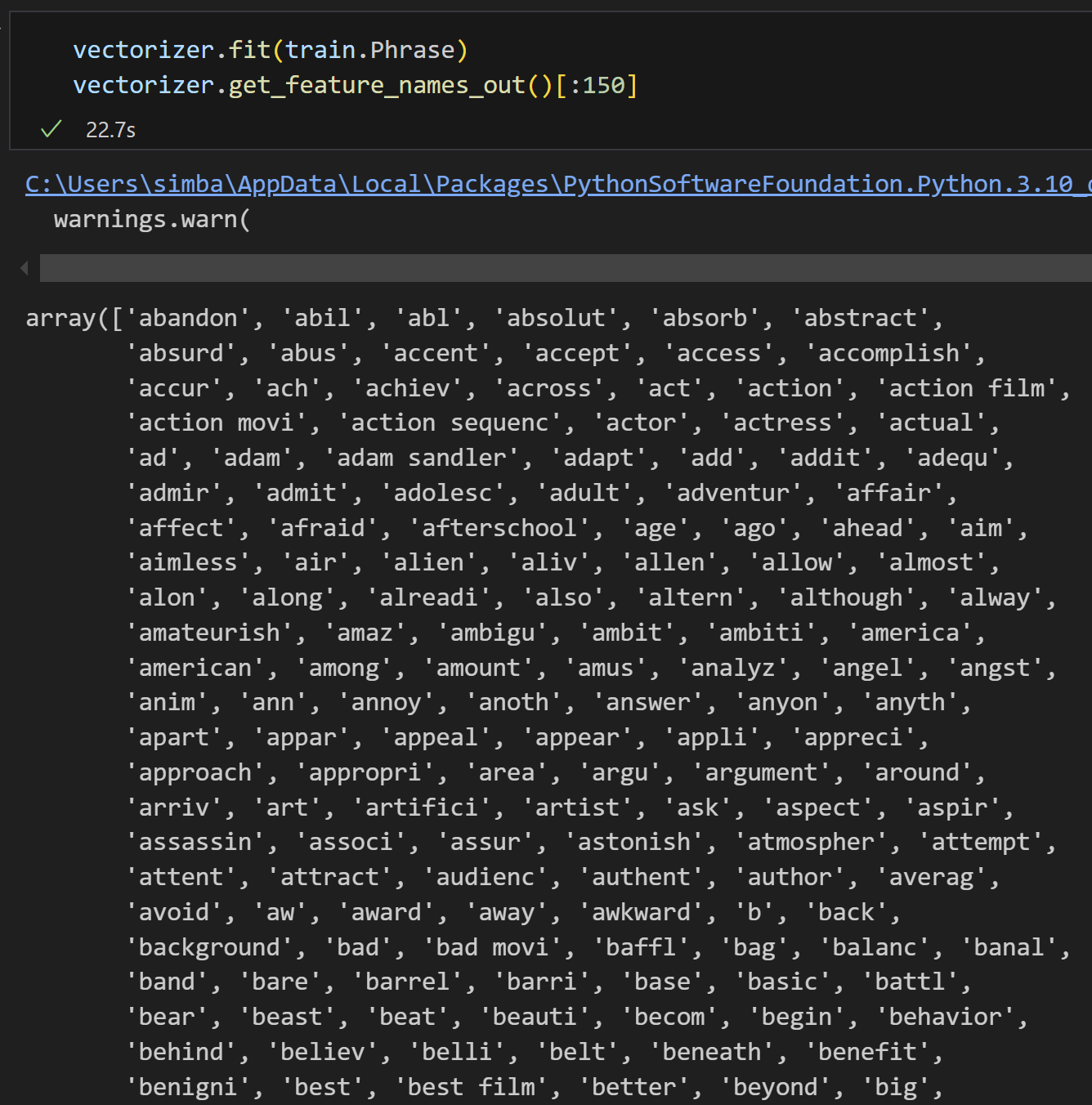
I applied the combined\_features function to the lemmatized dataset (phrasedocs\_2) to generate combined\_featuresets. This step integrates both bi-gram features and LIWC-based sentiment features.

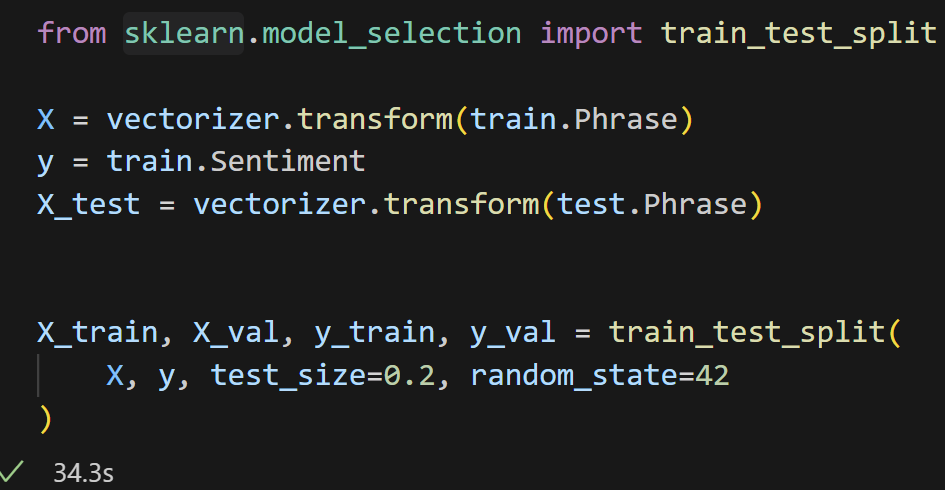
After training the Naive Bayes Classifier on the combined feature sets (combined\_featuresets), the model achieved an accuracy of 0.56325 (56.33%) on the test set. Although the combined feature set performed slightly better than some individual feature types, the improvement was not substantial. 

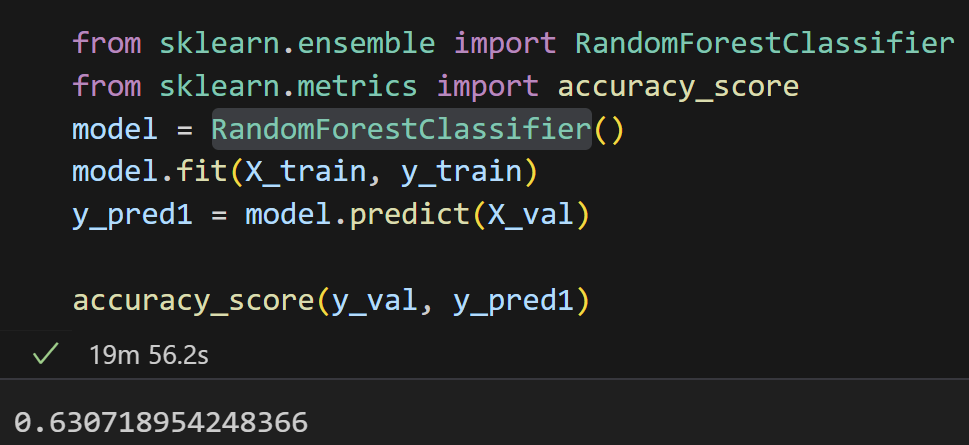
* 1. **Sklearn RandomForestClassifier**

After completing my project, I researched online to explore how others tackled this Kaggle competition. I came across a method that achieved the highest accuracy and noticed significant differences in their approach compared to mine. Intrigued by their methodology, I decided to adapt their code to my dataset and settings, making slight modifications where necessary.

I retained their data processing steps to understand their workflow and reasoning better. This exercise allowed me to gain insights into alternative approaches and refine my skills by learning how others process data and structure their solutions. By analyzing different perspectives, I aim to broaden my understanding and improve my problem-solving techniques. 



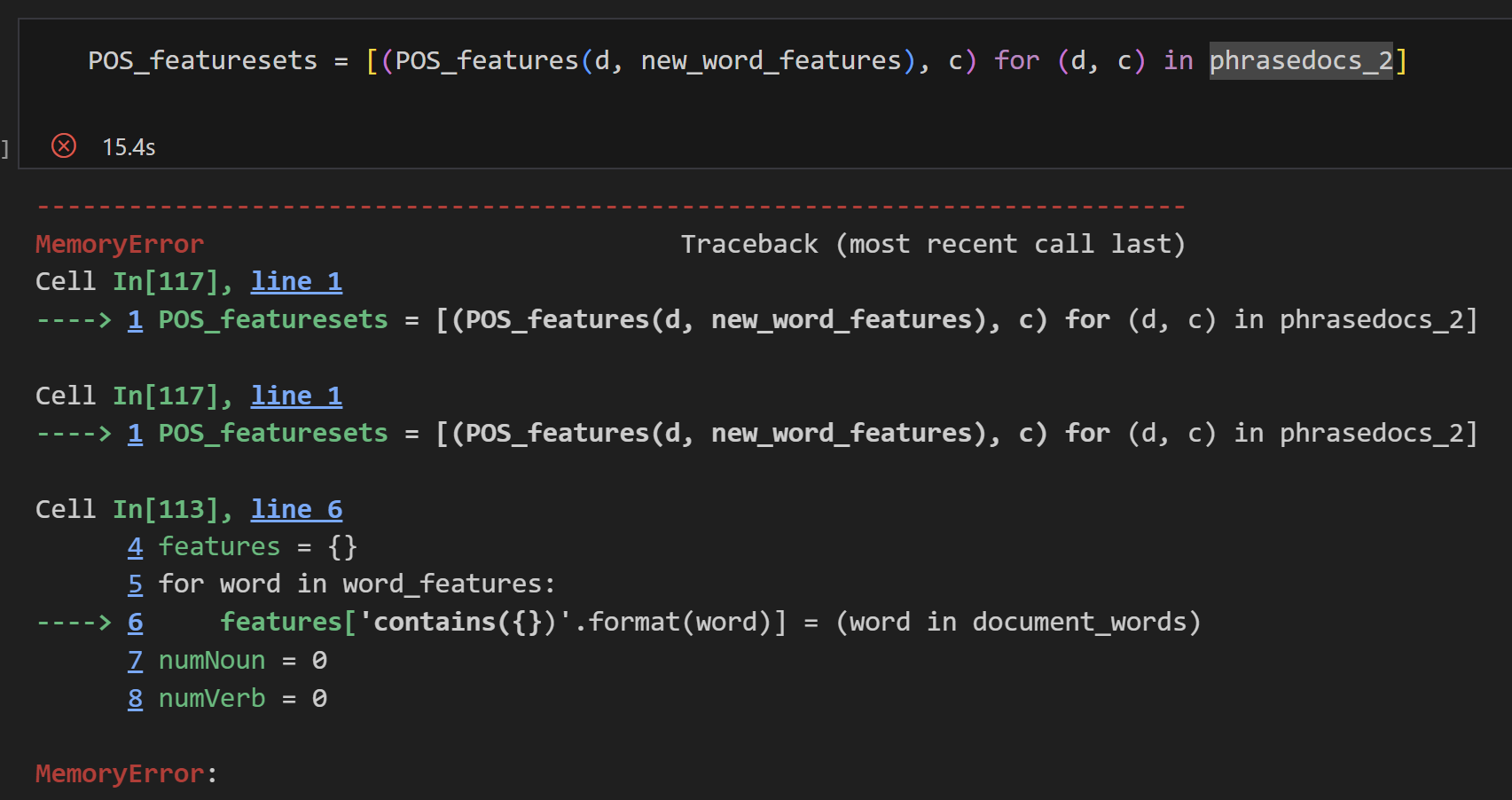


The Random Forest Classifier outperformed the Naive Bayes Classifier, achieving higher accuracy. While I may not fully understand all the intricate details of how Random Forest works, I know that it operates by creating and combining multiple decision trees to make predictions. This method appears to capture patterns in the data more effectively and might also be less prone to errors caused by individual trees making incorrect predictions. I can see why this approach is more robust in handling complex datasets compared to simpler models like Naive Bayes.

1. **Conclusion**

Throughout this project, I tested multiple models and feature engineering techniques to improve sentiment analysis accuracy. The Naive Bayes Classifier served as a baseline with an initial accuracy of 56.73%. By incorporating advanced features such as bi-grams, LIWC sentiment scores, and POS tags with combined features achieving an accuracy of 56.33%. The highest accuracy is Subjectivity one, 57.6%. The Random Forest Classifier outperformed the baseline, achieving a higher accuracy of 63.07%, highlighting the potential of ensemble methods in handling complex datasets. While the accuracy improvements across feature engineering techniques were moderate, each step provided valuable insights into the impact of different features. This numerical comparison shows that while simple models can provide a reasonable baseline, advanced models and carefully engineered features are essential for achieving better results in sentiment analysis tasks.

Before starting this final project, I spent a significant amount of time reviewing all the slides, labs, and homework assignments to refresh my memory. However, I faced challenges with my laptop's limited memory. Whenever I tried to use more complex functions or increase the dataset size, my laptop would unexpectedly crash. If I forgot to save my progress, I had to rethink my steps and re-run all the code, which was both time-consuming and frustrating. I initially thought I had saved enough time for this project after completing assignments from other courses and thoroughly reviewing the material. However, I quickly realized that the time was still not sufficient. Finally understand why professor kept telling us to start the final project as early as possible.



Even though I reviewed all the material, I realized that practical work is quite different from theoretical concepts when I started doing it hands-on. I should have begun earlier to identify which parts of the theory I wasn’t fully familiar with. Additionally, I didn’t spend much time combining different features to improve accuracy. Instead, my focus was on testing different methods and evaluating which features performed best.

When it came to combining features, I found it challenging because different features often have varying dataset requirements. This sometimes left me stuck while trying to integrate them effectively. Despite these challenges, I truly enjoyed working on this final project. I’ve learned a lot and feel inspired to continue studying NLP to gain a deeper understanding.

During the winter break, I plan to review what I’ve learned, work on improving the accuracy in this competition, and participate in another Kaggle NLP competition to solidify my understanding of NLP concepts. I also intend to study how others approach the competition, focusing on their methods and data processing techniques, to broaden my perspective and refine my skills.