Rating Prediction from Review

Marri Bharadwaj	B21CS045
Gagan Gandi	B21CS096

1. I have chosen the train, validation, and test split as 70:15:15.

2. I have done pre-processing(details in 3rd subtask) to the data and I found the TfIdf for the terms in the data and got the following results:

```
Top terms for reviewText based on average TF-IDF scores:
great: 0.0510
good: 0.0388
love: 0.0385
not: 0.0366
work: 0.0266
use: 0.0236
product: 0.0231
very: 0.0224
but: 0.0223
like: 0.0193
well: 0.0183
get: 0.0172
gift: 0.0170
one: 0.0162
nice: 0.0155
just: 0.0153
song: 0.0150
buy: 0.0140
price: 0.0137
time: 0.0134
```

```
Top terms for summary based on average TF-IDF scores:
star: 0.1804
five: 0.1729
great: 0.0391
good: 0.0294
four: 0.0279
love: 0.0236
work: 0.0190
not: 0.0165
product: 0.0155
well: 0.0112
three: 0.0106
nice: 0.0101
like: 0.0098
but: 0.0096
very: 0.0092
excellent: 0.0090
one: 0.0090
need: 0.0086
perfect: 0.0079
use: 0.0074
```

In the above images, I can see the *tf-idf* scores of each word. The greater the score, the more relevant the word is to the corpus.
The above two images are for the two columns '*reviewTex*t' and 'summary' in the data.

3. The text pre-processing pipeline I followed is:
   Lowercasing the text, handling contractions, removing special characters, tokenization, removing punctuation, removing stop words(not all but only those which are irrelevant because there are many stop words in English that are relevant in the case.), lemmatisation, and finally joining the tokens back.

4. I have trained the data for each of the given models and the detailed evaluation metrics for all the models I have provided in subtask 7.

5. As per the accuracies I got from the validation and test set random forest model works best for the data. The detailed accuracies and confusion matrix are in the subtask 7.

6. After making the problem a binary classification problem I got the following results:

```
Multinomial Naive Bayes Classifier:
Accuracy: 0.9372649113379903
Precision: 0.9511073965733389
Recall: 0.9832949308755761
F1-score: 0.9669333710967924
Confusion Matrix:
 [[ 149  351]
 [ 116 6828]]

Gaussian Naive Bayes Classifier:
Accuracy: 0.4747447608812466
Precision: 0.9381860196418256
Recall: 0.46774193548387094
F1-score: 0.6242552373630598
Confusion Matrix:
 [[ 286  214]
 [3696 3248]]

Decision Tree Classifier (Entropy):
Accuracy: 0.934309511015583
Precision: 0.9615329615329615
Recall: 0.9683179723502304
F1-score: 0.9649135394991748
Confusion Matrix:
 [[ 231  269]
 [ 220 6724]]

Decision Tree Classifier (Gini):
Accuracy: 0.9365932294465341
Precision: 0.9633447880870561
Recall: 0.9688940092165899
F1-score: 0.9661114302125214
Confusion Matrix:
 [[ 244  256]
 [ 216 6728]]
```

```
Random Forest Classifier (20 trees):
Accuracy: 0.9535196131112306
Precision: 0.9531593406593407
Recall: 0.9992799539170507
F1-score: 0.9756749156355456
Confusion Matrix:
 [[ 159  341]
 [   5 6939]]

Random Forest Classifier (50 trees):
Accuracy: 0.9552659860290167
Precision: 0.9543642611683849
Recall: 0.9998559907834101
F1-score: 0.9765806315493354
Confusion Matrix:
 [[ 168  332]
 [   1 6943]]

Random Forest Classifier (100 trees):
Accuracy: 0.9535196131112306
Precision: 0.952661909989023
Recall: 0.9998559907834101
F1-score: 0.9756885890949972
Confusion Matrix:
 [[ 155  345]
 [   1 6943]]
```

7. Here are the detailed evaluation metrics of the models I got:

```
Multinomial Naive Bayes:
Accuracy: 0.7368350349274584
Precision: 0.7053971483595791
Recall: 0.7368350349274584
F1 Score: 0.7022446699970953
Confusion Matrix:
[[  67    4   23   45  160]
 [   4    9   21   57  110]
 [   4    2  144  132  260]
 [   9    1   30  339  741]
 [  11    2   33  310 4926]]

Gaussian Naive Bayes:
Accuracy: 0.2435518538420204
Precision: 0.56749892168377
Recall: 0.2435518538420204
F1 Score: 0.3048748150092578
Confusion Matrix:
[[ 135   23   27   53   61]
 [  68   37   22   34   40]
 [ 152   39  171   73  107]
 [ 383   89  131  266  251]
 [2448  381  600  649 1204]]
```

```
Decision Tree (Entropy):
Accuracy: 0.7673293927995701
Precision: 0.761193151493996
Recall: 0.7673293927995701
F1 Score: 0.763799974715004
Confusion Matrix:
[[ 122   20   36   33   88]
 [  13   73   19   42   54]
 [  17   27  267   82  149]
 [  27   32  101  543  417]
 [  56   53  123  343 4707]]

Decision Tree (Gini):
Accuracy: 0.773374529822676
Precision: 0.764732432974373
Recall: 0.773374529822676
F1 Score: 0.7681792774901642
Confusion Matrix:
[[ 132   17   20   32   98]
 [  11   74   19   29   68]
 [  14   21  269   80  158]
 [  23   24   71  558  444]
 [  42   58  119  339 4724]]

Random Forest (20 trees):
Accuracy: 0.8069586243954863
Precision: 0.8110726770636051
Recall: 0.8069586243954863
F1 Score: 0.7781608168513955
Confusion Matrix:
[[  96    1    4    8  190]
 [   3   61    5   12  120]
 [   1    1  212   44  284]
 [   1    1   13  452  653]
 [   2    1   19   74 5186]]

Random Forest (50 trees):
Accuracy: 0.8080333154218162
Precision: 0.8278684623139689
Recall: 0.8080333154218162
F1 Score: 0.7752270636064788
Confusion Matrix:
[[  84    1    0    8  206]
 [   1   62    0    7  131]
 [   0    0  207   19  316]
 [   0    2    9  421  688]
 [   0    0    1   40 5241]]
```

```
Random Forest (100 trees):
Accuracy: 0.8074959699086512
Precision: 0.8289003998312873
Recall: 0.8074959699086512
F1 Score: 0.7740065697253204
Confusion Matrix:
[[  88    1    0    5  205]
 [   2   62    1    5  131]
 [   0    0  204   25  313]
 [   0    0    6  409  705]
 [   0    0    2   32 5248]]
```

9. I can see that Gaussian naive Bayes has the lowest accuracy which might be because Gaussian Naive Bayes assumes that features follow a Gaussian distribution, which may not hold true for all datasets.
Random Forest is an ensemble learning method that combines multiple decision trees to improve performance and reduce overfitting. By aggregating the predictions of multiple trees, Random Forest can produce more accurate and stable predictions.

Task- I

Part-of-Speech (POS) taggers and Named Entity Recognition (NER) taggers are essential components of natural language processing (NLP) systems.

POS taggers assign grammatical tags to each word in a text, indicating the word's syntactic category such as noun, verb, adjective, etc. These tags help in understanding the structure and meaning of sentences, aiding in tasks like parsing, language understanding, and machine translation.

On the other hand, NER taggers identify and classify named entities in text, such as persons, organisations, locations, dates, and more. These entities provide crucial information for tasks like information extraction, document categorization, and sentiment analysis. NER taggers segment and label these entities, enabling systems to extract relevant information from unstructured text data.

In the first task, I performed the Comparative analysis of the performance of pre-trained linguistic parsers and Named Entity Recognition (NER) models on given English datasets using standard metrics.

1. SpaCy-

   SpaCy is a leading open-source library for natural language processing (NLP) in Python. It provides efficient and accurate implementations of various NLP components, including tokenization, POS tagging, NER, dependency parsing, and more. SpaCy's key features include pre-trained models for multiple languages, easy-to-use APIs, and high performance, making it a popular choice for building NLP pipelines and applications. Its advanced capabilities, such as entity linking, word vectors, and custom rule-based matching, make it suitable for a wide range of tasks, from simple text analysis to complex information extraction and semantic understanding.

   NLTK-

   Natural Language Toolkit is a widely used library for natural language processing (NLP) in Python. It offers a comprehensive suite of tools and resources for various NLP tasks, including tokenization, POS tagging, NER, sentiment analysis, and more. NLTK provides extensive collections of lexical resources, corpora, and pre-trained models, facilitating research and development in NLP. Its modular design allows users to easily combine different components to build custom NLP pipelines tailored to specific tasks. Additionally, NLTK offers a user-friendly interface and detailed documentation, making it accessible to both beginners and experts in the field. With its rich set of functionalities and wide community support, NLTK continues to be a go-to choice for NLP tasks across academia and industry.

   StanforNER-

   Stanford NER (Named Entity Recognizer) is a widely used tool for identifying and classifying named entities in text. Developed by the Stanford NLP Group, it offers robust performance and support for multiple languages. Stanford NER utilises machine learning techniques, such as conditional random fields (CRFs) and maximum entropy classifiers, to recognize entities like persons, organisations, locations, dates, and more. It provides pre-trained models for

various domains and allows users to train custom models on their data for specific applications. Stanford NER's accuracy, scalability, and flexibility make it a popular choice for NER tasks in academia, industry, and research. Its reliable performance and comprehensive documentation make it a valuable tool for NLP practitioners seeking accurate entity recognition capabilities.

I performed preprocessing by lemmitisation and removed irrelevant column as as shown below-

	Word	POS	Tag
0	['Thousands', 'of', 'demonstrators', 'have', '...	['NNS', 'IN', 'NNS', 'VBP', 'VBN', 'IN', 'NNP'...	['O', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'O', '...
1	['Iranian', 'officials', 'say', 'they', 'expec...	['JJ', 'NNS', 'VBP', 'PRP', 'VBP', 'TO', 'VB',...	['B-gpe', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '...
2	['Helicopter', 'gunships', 'Saturday', 'pounde...	['NN', 'NNS', 'NNP', 'VBD', 'JJ', 'NNS', 'IN',...	['O', 'O', 'B-tim', 'O', 'O', 'O', 'O', 'O', '...
3	['They', 'left', 'after', 'a', 'tense', 'hour-...	['PRP', 'VBD', 'IN', 'DT', 'NN', 'JJ', 'NN', '...	['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', ...
4	['U.N.', 'relief', 'coordinator', 'Jan', 'Egel...	['NNP', 'NN', 'NN', 'NNP', 'NNP', 'VBD', 'NNP'...	['B-geo', 'O', 'O', 'B-per', 'I-per', 'O', 'B-...

The WORD column represents the pre-tokenised sentences, followed by the Parts-of-Speech and Tags attributed in the dataset.

I then performed Exploratory Data Analysis to get an overall description of the data as follows-

```
Number of rows: 47959
Number of columns: 3
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47959 entries, 0 to 47958
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Word    47959 non-null  object
 1   POS     47959 non-null  object
 2   Tag     47959 non-null  object
dtypes: object(3)
memory usage: 1.1+ MB
None
                                                  Word  \
count                                            47959   
unique                                           47575   
top     ['VOA', "'s", 'Mil', 'Arcega', 'reports', '.']   
freq                                                17   

                                            POS  \
count                                     47959   
unique                                    47214   
top     ['NNP', 'POS', 'NNP', 'NNP', 'VBZ', '.']   
freq                                         39   

                                                      Tag  
count                                               47959  
unique                                              33318  
top     ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', ...  
freq                                                  450  
```

I explored the distribution of POS and NER tags as shown below-

```
POS tag distribution:
 ['NNP', 'POS', 'NNP', 'NNP', 'VBZ', '.']
['NNP', 'NNP', 'VBZ', '.']
['NNP', 'POS', 'NNP', 'NNP', 'VBZ', 'RBR', '.']
['NNP', 'VBZ', 'DT', 'NN', '.']
['DT', 'NN', 'VBZ', 'VBN', 'NN', 'IN', 'DT', 'NN', '.']

['WRB', 'DT', 'NN', 'VBD', 'IN', 'PRP$', 'NN', ',', 'DT', 'NN', 'VBD', 'PRP$', 'NN', 'IN', 'IN', 'DT', 'NNS', 'CC', 'VBD', 'IN', 'PRP', 'MD', 'VB', 'PRP$', 'NNS', 'IN', 'PRP', 'MD', '\
['CD', 'IN', 'DT', 'NNS', 'VBD', ',', '``', 'PRP', 'MD', 'RB', 'RB', 'VB', 'VBN', 'IN', 'PRP$', 'NN', 'IN', 'DT', 'NN', 'IN', 'VB', 'DT', 'NN', 'IN', 'PRP', 'IN', 'PRP$', 'NNS', '.', '
['CD', 'JJ', 'NN', 'POS', 'NN', 'DT', 'NN', 'VBD', 'VBG', 'IN', 'DT', 'NN', 'IN', 'PRP', 'VBD', 'TO', 'DT', 'NN', 'IN', 'NNS', 'RB', 'VBG', 'IN', 'DT', 'NN', 'WDT', 'VBD', 'VBN', 'VBN'
['``', 'RB', 'DT', 'NN', 'TO', 'VB', 'PRP$', 'NN', ',', '``', 'VBD', 'PRP', '.']
['DT', 'NNP', 'NNP', 'VBZ', 'VBG', 'DT', 'NN', 'IN', 'JJ', 'NNS', 'TO', 'VB', 'NN', 'CC', 'NN', 'NNS', 'IN', 'JJ', 'NNP', ',', 'VBG', 'DT', 'NN', 'VBP', '``', 'IN', 'PRP$', 'NN', 'IN',
Name: POS, Length: 47214, dtype: int64
Named entity tag distribution:
 ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']                                                              45(
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']                                                430
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']                                                     416
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']                                           401
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']                                                                    401
                                                                                                                     ...
['B-gpe', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-per', 'O', 'O', 'O', 'O']                          1
['O', 'O', 'B-tim', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']          1
['O', 'O', 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']                          1
['O', 'B-tim', 'I-tim', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O', 'B-tim', 'O']                          1
['O', 'B-org', 'I-org', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-tim', 'I-tim', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']      1
Name: Tag, Length: 33318, dtype: int64
```

I then utilised the first POS and NER tagger- ScaPy. It predicted the tags of our tokenised datapoints as follows-

	Word	POS	Tag	Predicted_POS	Predicted_NER
0	['Thousands', 'of', 'demonstrators', 'have', '...	['NNS', 'IN', 'NNS', 'VBP', 'VBN', 'IN', 'NNP'...	['O', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'O', '...	[X, X, NOUN, NUM, PUNCT, PUNCT, ADP, NUM, PUNC...	[GPE, GPE, NORP]
1	['Iranian', 'officials', 'say', 'they', 'expec...	['JJ', 'NNS', 'VBP', 'PRP', 'VBP', 'TO', 'VB',...	['B-gpe', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '...	[X, PUNCT, ADJ, PROPN, PUNCT, PUNCT, NOUN, PAR...	[NORP, DATE]
2	['Helicopter', 'gunships', 'Saturday', 'pounde...	['NN', 'NNS', 'NNP', 'VBD', 'JJ', 'NNS', 'IN',...	['O', 'O', 'B-tim', 'O', 'O', 'O', 'O', 'O', '...	[X, PUNCT, PROPN, PUNCT, PUNCT, PUNCT, NOUN, P...	[DATE, PERSON, ORG, LOC, PERSON]
3	['They', 'left', 'after', 'a', 'tense', 'hour-...	['PRP', 'VBD', 'IN', 'DT', 'NN', 'JJ', 'NN', '...	['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', ...	[X, PUNCT, PRON, VERB, PUNCT, PUNCT, VERB, PUN...	[TIME]
4	['U.N.', 'relief', 'coordinator', 'Jan', 'Egel...	['NNP', 'NN', 'NN', 'NNP', 'NNP', 'VBD', 'NNP'...	['B-geo', 'O', 'O', 'B-per', 'I-per', 'O', 'B-...	[X, PUNCT, PROPN, PROPN, PUNCT, PUNCT, NOUN, P...	[ORG, ORG, DATE, GPE, NORP, NORP, GPE]

Here “Predicted_POS” and “Predicted “NER” represents that of ScaPy.

The mapping of the outputted tags and actual tags in the POS and Tag columns is shown below-

```
Missing values in POS tags: 0
Unique values in POS tags: ["['NNS', 'IN', 'NNS', 'VBP', 'VBN', 'IN', 'NNP', 'TO', 'VB', 'DT', 'NN', 'IN', 'NNP', 'CC', 'VB', 'DT', 'NN', 'IN', 'JJ', 'NNS', 'IN', 'DT', 'NN', '.']"
 "['JJ', 'NNS', 'VBP', 'PRP', 'VBP', 'TO', 'VB', 'NN', 'TO', 'JJ', 'JJ', 'NNS', 'IN', 'DT', 'NN', 'NNP', ',', 'IN', 'DT', 'NNP', 'NN', 'NN', 'VBZ', 'VBG', '.']"
 "['NN', 'NNS', 'NNP', 'VBD', 'JJ', 'NNS', 'IN', 'DT', 'NNP', 'JJ', 'NN', ',', 'WRB', 'JJ', 'NNP', 'NNS', 'VBP', 'VBN', 'TO', 'VB', 'VBN', 'TO', 'VB', 'DT', 'JJR', 'JJ', 'NN', 'IN', ':
 ...
 "['VBG', 'NNP', 'POS', 'JJ', 'NNP', 'CD', 'NNS', ',', 'NNS', 'NNS', 'VBD', 'NNS', 'IN', 'NNS', 'VBD', 'VBN', 'IN', 'NNS', 'IN', 'NN', 'NNS', 'IN', 'NN', ',', 'JJ', 'NNS', '.']"
 "['IN', 'RB', ',', 'NNS', 'VBP', 'VBN', 'JJ', 'NNS', 'IN', 'DT', 'VBN', 'CC', 'VBN', 'TO', 'VB', 'JJ', 'NNS', 'IN', 'DT', 'NN', '.']"
 "['DT', 'NNP', 'NNP', 'VBZ', 'VBG', 'DT', 'NN', 'IN', 'JJ', 'NNS', 'TO', 'VB', 'NN', 'CC', 'NN', 'NNS', 'IN', 'JJ', 'NNP', ',', 'VBG', 'DT', 'NN', 'VBP', '``', 'IN', 'PRP$', 'NN', 'I
Example 1
Predicted POS: ['X', 'X', 'NOUN', 'NUM', 'PUNCT', 'PUNCT', 'ADP', 'NUM', 'PUNCT', 'PUNCT', 'NOUN', 'PUNCT', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT', 'PUNCT
Actual POS: ['NNS', 'IN', 'NNS', 'VBP', 'VBN', 'IN', 'NNP', 'TO', 'VB', 'DT', 'NN', 'IN', 'NNP', 'CC', 'VB', 'DT', 'NN', 'IN', 'JJ', 'NNS', 'IN', 'DT', 'NN', '.']

Example 2
Predicted POS: ['X', 'PUNCT', 'ADJ', 'PROPN', 'PUNCT', 'PUNCT', 'NOUN', 'PART', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT', 'PUNCT', 'PUNCT', 'PRON', 'VERB', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT',
Actual POS: ['JJ', 'NNS', 'VBP', 'PRP', 'VBP', 'TO', 'VB', 'NN', 'TO', 'JJ', 'JJ', 'NNS', 'IN', 'DT', 'NN', 'NNP', ',', 'IN', 'DT', 'NNP', 'NN', 'NN', 'VBZ', 'VBG', '.']

Example 3
Predicted POS: ['X', 'PUNCT', 'PROPN', 'PUNCT', 'PUNCT', 'PUNCT', 'NOUN', 'PROPN', 'PUNCT', 'PUNCT', 'PROPN', 'PUNCT', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT', 'PUNCT', 'PUNCT', 'ADJ', 'PUN
Actual POS: ['NN', 'NNS', 'NNP', 'VBD', 'JJ', 'NNS', 'IN', 'DT', 'NNP', 'JJ', 'NN', ',', 'WRB', 'JJ', 'NNP', 'NNS', 'VBP', 'VBN', 'TO', 'VB', 'VBN', 'TO', 'VB', 'DT', 'JJR', 'JJ', 'NN

Example 4
Predicted POS: ['X', 'PUNCT', 'PRON', 'VERB', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT', 'PUNCT', 'PUNCT', 'ADP', 'PUNCT', 'PUNCT', 'PUNCT', 'PRON', 'NUM', 'PUNCT', 'PUNCT', 'ADJ', 'PUNCT', 'P
Actual POS: ['PRP', 'VBD', 'IN', 'DT', 'NN', 'JJ', 'NN', 'IN', 'NN', 'NNS', '.']

Example 5
Predicted POS: ['X', 'PUNCT', 'PROPN', 'PROPN', 'PUNCT', 'PUNCT', 'NOUN', 'PUNCT', 'PUNCT', 'PUNCT', 'NOUN', 'PUNCT', 'PUNCT', 'PUNCT', 'PROPN', 'PROPN', 'PUNCT', 'PUNCT', 'PROPN', 'PU
Actual POS: ['NNP', 'NN', 'NN', 'NNP', 'NNP', 'VBD', 'NNP', ',', 'NNP', ',', 'JJ', 'CC', 'JJ', 'JJ', 'NNS', 'VBP', 'VBG', 'RP', 'NN', 'CC', 'NNS', 'TO', 'VB', 'NNS', 'IN', 'JJ', 'NNP',

Example 6
Predicted POS: ['X', 'PUNCT', 'PROPN', 'PROPN', 'PUNCT', 'PUNCT', 'PROPN', 'PUNCT', 'PUNCT', 'PUNCT', 'VERB', 'PUNCT', 'PUNCT', 'PUNCT', 'DET', 'PUNCT', 'PUNCT', 'PUNCT', 'ADJ', 'PUNCT
Actual POS: ['NNP', 'NNP', 'VBD', 'DT', 'JJS', 'NNS', 'VBP', 'CD', 'CD', 'NNS', 'VBP', 'IN', 'NN', 'IN', 'NN', 'NN', ':', 'IN', 'DT', 'NN', 'JJS', 'IN', 'NNP', ',', 'NNP', 'NNP', ',',
```

Similar approach have I done for the second tagger- NLTK or Natural Language Toolkit:

	Word	POS	Tag	Predicted_POS	Predicted_NER	Predicted_POS_NLTK	Predicted_NER_NLTK
0	['Thousands', 'of', 'demonstrators', 'have', '...	['NNS', 'IN', 'NNS', 'VBP', 'VBN', 'IN', 'NNP'...	['O', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'O', '...	[X, X, NOUN, NUM, PUNCT, PUNCT, ADP, NUM, PUNC...	[GPE, GPE, NORP]	[JJ, NNS, POS, ,, NNP, POS, ,, NNS, POS, ,, VB...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
1	['Iranian', 'officials', 'say', 'they', 'expec...	['JJ', 'NNS', 'VBP', 'PRP', 'VBP', 'TO', 'VB',...	['B-gpe', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '...	[X, PUNCT, ADJ, PROPN, PUNCT, PUNCT, NOUN, PAR...	[NORP, DATE]	[JJ, JJ, POS, ,, NNS, POS, ,, NNP, POS, ,, NNP...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
2	['Helicopter', 'gunships', 'Saturday', 'pounde...	['NN', 'NNS', 'NNP', 'VBD', 'JJ', 'NNS', 'IN',...	['O', 'O', 'B-tim', 'O', 'O', 'O', 'O', 'O', '...	[X, PUNCT, PROPN, PUNCT, PUNCT, PUNCT, NOUN, P...	[DATE, PERSON, ORG, LOC, PERSON]	[RB, NNP, POS, ,, NNS, POS, ,, ", ", ,, VBD,...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
3	['They', 'left', 'after', 'a', 'tense', 'hour-...	['PRP', 'VBD', 'IN', 'DT', 'NN', 'JJ', 'NN', '...	['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', ...	[X, PUNCT, PRON, VERB, PUNCT, PUNCT, VERB, PUN...	[TIME]	[NN, ", ", ,, ", ", ,, NNP, POS, ,, ", DT...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
4	['U.N.', 'relief', 'coordinator', 'Jan', 'Egel...	['NNP', 'NN', 'NN', 'NNP', 'NNP', 'VBD', 'NNP'...	['B-geo', 'O', 'O', 'B-per', 'I-per', 'O', 'B-...	[X, PUNCT, PROPN, PROPN, PUNCT, PUNCT, NOUN, P...	[ORG, ORG, DATE, GPE, NORP, NORP, GPE]	[NN, ", NNP, ., ", ,, ", ", ,, NNP, POS, ,...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...

Here, the last two columns are the predicted POS and Tags of the NLTK Model.

Similar for StanfordNER can be viewed in the python notebook attached in the zip folder.

2. The text processing pipeline employed above involves several steps to preprocess and analyse the text data using natural language processing (NLP) techniques. Let's break down the pipeline into its components:
   Loading the Dataset: The pipeline starts by loading the dataset containing text data. This dataset could be in various formats such as CSV, Excel, or plain text files.

- Preprocessing the Dataset: After loading the dataset, preprocessing steps are applied to clean and prepare the text data for analysis. This typically includes tasks such as:
- Tokenisation: Splitting the text into individual tokens or words.
- Lemmatisation or stemming: Reducing words to their base or root form to normalise the text. This helps in reducing the vocabulary size and improving model performance.
- Text Analysis with NLP Libraries:
- SpaCy Analysis: In this step, SpaCy, a popular NLP library, is used for text analysis tasks such as POS tagging and NER (Named Entity Recognition). SpaCy provides pre-trained models and functionalities to perform these tasks efficiently.
- NLTK Analysis: Similarly, NLTK (Natural Language Toolkit), another NLP library, is used for text analysis. NLTK provides various tools and modules for tasks like POS tagging, tokenization, and named entity recognition.

Predicting POS Tags and NER:

- For SpaCy, the pre-trained models are used to predict POS tags and NER tags for each word in the text. These predictions are then stored in separate columns in the dataset.
- For NLTK, POS tags and NER tags are predicted using NLTK's default POS tagger and a simple rule-based NER approach based on named entity chunking.

Comparing Predictions with Actual Tags:

- Once the predictions are made, the pipeline compares the predicted POS tags and NER tags with the actual tags provided in the dataset.
- The predictions are mapped or converted to match the format of the actual tags, and then accuracy and confusion matrix metrics are calculated to evaluate the performance of the models.

Analysis and Interpretation:

- Finally, the pipeline involves analysing the results, identifying any discrepancies or patterns between the predicted and actual tags, and interpreting the performance of the models. This may involve visualising the results, exploring individual predictions, and understanding the strengths and limitations of the NLP models employed.

5. The tags provided by the models differ from those specified in the NER_Dataset hence the following mapping had to be employed-

```
def map_spacy_ner_tags(tag):
    mapping = {
        "B-art": "ART",
        "B-eve": "EVENT",
        "B-geo": "GPE",
        "B-gpe": "GPE",
        "B-nat": "NORP",
        "B-org": "ORG",
        "B-per": "PERSON",
        "B-tim": "TIME",
        "I-art": "ART",
        "I-eve": "EVENT",
        "I-geo": "GPE",
        "I-gpe": "GPE",
        "I-nat": "NORP",
        "I-org": "ORG",
        "I-per": "PERSON",
        "I-tim": "TIME",
        "O": ""
    }
```

I faced the important challenge of acquiring the trained model especially for the last model of StanfordNER unfortunately inspite of immense internet browsing.