**Idea of the whole project:**

First, I will import necessary libraries for NLP and define some functions for processing. Then I will load 44 MOUs file and extract only compensation of each MOU except the first MOU. I apply NLP methods to clean up data that I have extracted from raw text and prepare it for training process. To prepare for model classification I split data into training and testing data with appropriate format. During training process, training data was fed into classification model and then the model was evaluated with the testing data.

In next step, I process the text from MOU1 compensation section and use it as unlabeled data to apply my classification model on. I used my model to chunk the text from MOU1 compensation and then extract bonus information. I then stored the extracted info in SQL database. Finally, I created an application to retrieve info from that SQL database and display the result base on the input that users type in.

**1. Import necessary libraries for Natural Language Processing**

We need to import necessary libraries and tools to process text data. NLP libraries include nltk, PyPDF2, docx. Tools from nltk libraries are word\_tokenize, pos\_tag, conlltags2tree tree2conlltags, ChunkParserI, TaggerI, NaiveBayesClassifier, RegexpParser, grammar, parse, WordNetLemmatizer, and Tree. General librabries to manipulate data structure and objects and database are Numpy, Pandas, Random, csv, Sqlite3, pprint.

**2. Self-defined functions**

**2.1 pdf\_extract() function**

This function is created to process raw input pdf file. The function will extract text from pdf format within the range of pages provided. It will take pdf file, the first page number and the last page number as input parameters. The function will return a list of strings, each string is a document in the input corpus.

**2.2 getText() function**

This function is used for docx file text extraction. Text will be read and combined together. This function will return the full text.

**2.3 text\_process() function**

This function is created to clean up text data. The function will tokenize each doc of input text corpus, lemmatize words, remove redundant functuation except $ and %, exclude words whose length is 1 and segmentize sentences by detect full stop punctuation. This function will return the list of tokenized sentences.

**2.4 df\_pos() function**

This function is to create a data frame to restore words and their POS tags. This function takes tree form objects with POS tags as its input parameters. It will return a data frame with three columns: sentence number, words in each sentence and corresponding POS tags.

**2.5 df\_iob() function**

This function is identical to df\_pos() function except adding IOB tags. It takes tree objects with IOB tags as input parameters. It will then return a data frame with four columns: sentence number, words with corresponding POS and IOB tags.

**2.6 flatten\_childtrees() and flatten\_deeptree() functions**

These two punctions works together to flatten deep trees whose heights are more than 3. They take tree objects as their input parameters. Flatten trees are compatible with IOB tags while nested trees are not. These functions will make the training data suitable for chunker training.

**2.7 ChunkTagger Class and ChunkClassifier Class**

These two classes are used for chunking sentences with classifier-based method.

ChunkTagger class will create a tagger which extract features of each sentences using self-defined feature extractor chunk\_features() function, train the classifier with NaiveBayesClassifier and predict the tag with tag() method

ChunkClassifier class will turn the tagger into a chunker. It will map the chunk tree to tagged sentence and transform triplets into pairs format which are compatible with tagger interface. It then uses parse() method to convert the result from tagger back to chunk trees

**2.8 chunk\_features() function**

This function is a feature extractor which will extract features from tokenized sentences. The extracted features include the target word and its pos tag, 2 words before the target word and their pos tags, 2 words after the target word and their pos tags, the pair of pos tags of target word and its predecessor, the pair of pos tags of target word and its seccessor.

**3. Process raw data**

All text related to compensation of 44 MOUs will be extracted to create traning and testing data. The first MOU is used as the unlabel text for tagging and chunking, other MOUs is used as the training and testing data to build a classifier-based chunker.

Compensation information from MOUs will be read from pdf file by pages using pdf\_extract() function. It then will be cleaned up using process\_text() function and tagged with POS tag using pos\_tag() function from nltk. After that, it will be exported to csv file for manual POS tags corrrection to increase the accuracy.

Correct POS tags sentences will be imported and chunked with regular expression based approach with grammar consist of Noun Phrases, Verb Phrases and Numbers rules. Since tagged sentences are now deep trees, it is essential to apply flatten\_deeptree() over them to transform these sentences into format that is compatible for classification

**4. Create training and testing data, set up classification model and feed the data into model**

Train and test data from the processing step will be randomly split into training set and testing set with the ration of 80:20. random\_sample() function from random is used to perform this task

After data had been processed and two set of data had been created, train set was feed into ChunkClassifier() to create a chunker. The chunker then was used for evaluation with test data.

**5. Use classification model to classify unlabeled data set**

Text from MOU1\_Compensation docx file was read using getText() function. Text then was separated by paragraphs, and article names are excluded and archived in another list for later use. From here the text process was similar to the previous text process for creating train and test sets including POS tagging, exporting to csv for POS tag correction and importing back for chunk classification.

Then, the chunker from training process was applied on MO1 text. The parse() method from ChunkClassifier was used for predicting the chunk types of this unlabeled dataset. The chunker returned a list of chunked sentences.

**6. Perform deep chunk onto classifed data set to extract information**

After MOU1 text had been chunked, another direct chunk process with grammar was created by regular expression rules to perform a deep chunk for bonus information extraction. The rule created Clauses and Number Phrases rules based on NP, VP, NUM predicted from model. Clauses chunked the employee information while the Number Phrases chunked the percentage or amount related to bonus information

Bonus information will be extracted by detected the existing of Clauses and Number Phrases on each sentence. If the sentence includes both Clauses and Number Phrases, it will store the first Clause and all Number Phrases into a dataframe. The dataframe consists of three columns: Section information from article name archived earlier, Employee information, and Amount of bonus information with percentage or amount

**7. Create SQL database to restore extracted information**

To restore the dataframe to a SQL database, the dataframe first export to csv file. Sqlite3 library are ultilized to create a SQL connection, execute SQL queries to create a new database with bonus\_table, and insert values from csv file. After that, The SQL database will be call out for information extraction.

**8. Establish an application to extract information from database using SQL query**

An application was created to extract bonus information from the SQL database made from previous step.

To make this application work, an SQL query need to be created based upon the input that users type in. To transform from human language to SQL language, a CFG structure with lexical productions was made. From nltk, grammar module conducted the grammar from the string and parse module read the grammar and created a parser.

The application consists of three processes. The first step is to generate a user interface that will ask a user to type in the section which the bonus information will be extracted. The second step is to collect information from the first step and use the parser just built to translate from a string of English to a SQL query. The final step is to call out the bonus database generated early using Sqlite3 and execute the SQL query to yield the result.