1. Question 1

There are two main functions that need completing:

a. **‘parse_data_line’**: the function takes one argument *data_line* as a single input, and returns a tuple of *label* and *text*. In this case, *data_line* should be each row from the file, which is equivalent to the ‘line’ in loop <*for line in reade*r> in ‘*load_data*’ function. Thus, to extract *label* and *text* data from a list, we simply subset it with index 1 and 2 in turn.
b. **‘pre_process’**: the function takes one argument *text* as an input and returns *a list of tokens*. The function is then implemented in a function called ‘*split_and_preprocess_data*’, where the input of ‘*pre_process*’ function follows string type. Hence, to split a *text* into tokens, the string split method with default separator (any whitespace) is used.

Moreover, the ‘*load_data*’ function needs argument “*encoding = ‘utf-8’*” in the open method to avoid any unnecessary errors.

2. Question 2

In this question, there is only a function: ‘**to_feature_vector**’. This takes one argument *tokens* as input and aims to return *vocabulary*. This can be solved by checking whether each *token* from the text is within a *vocabulary* dictionary created at first – this will be initially empty and gradually added when looping through the *tokens* input; if not, add that *token* as a new key with value 1; otherwise, update that *token* with <new value = current + 1>.

3. Question 3

In question 3, there is only one main function ‘**cross_validate**’ to be finished. This function takes *dataset* and *folds* as arguments, then returns the *average model performance* after folding. The idea is to divide the input *dataset* into 2 sets: training and validation. The validation set is decided by the fold_size and the train set is defined as <dataset – validation set>. Then, the classifier is trained with the train set and used to predict on validation. After folding k times (k-time iteration), the performance of each loop is stored and used to calculate *average score* (precision/recall/f1[1] & accuracy[2]).

4. Question 4

The label order in ‘**confusion_matrix_heatmap**’ is FAKE – REAL, where FAKE is a positive label and REAL is a negative one. The ratio of false positive (FP) cases and false negative (FN) ones is quite balance, though a bit inclined to more FN.

In the *error analysis* on a simple train-test split of training data, the classifier predicts the REAL labels better than FAKE (f1 score: 0.62 and 0.5 respectively). Looking at the FP and FN, there are some assumptions for model misclassification:

a. There are a mixture of capital and lowercase words, which may cause confusion.
b. There are many common words (a, an, the,…)
c. Some tokens are glued to a punctuation due to the str.split() method using any whitespace as separator.

5. Question 5

The idea is to individually test each modification in a greedy way (keep what is good for score and use it for further test). Below is what I have tried:

Table 1: Different pre-process method testing and tunning model hyperparameter (question 5)

Model modification	**Details**	**Impact**	**Decision**	**Updated method for better model**
‘pre_process’ function: use word_tokenize method	Replace the traditional tokenizing string split, as the new one returns cleaner tokens, splitting punctuation	No improvement	Keep the str.split()	No change
‘pre_process’ function: remove punctuation	Use str.isalpha()	No improvement	Do not remove punctuation	No change

(1) Precision/recall/f1 score are achieved by implementing the precision_recall_fscore_support method from sklearn
(2) Accuracy is defined by a function called ‘accuracy_calculate’: count how many cases where predicted class y_pred = true class y_true, then dividing it by total observations of true labels

‘pre_process’ function: remove stopwords	Get rid of too common words, such as a/an/…	Improve all scores	Remove stop-words	Remove stop-words(1)
‘pre_process’ function: normalize tokens	Convert words into lowercase with str.lower()	No improvement	Do not normalize	(1)
‘pre_process’ function: lemmatize tokens	Use WordNetLemmatizer from nltk package	No improvement	Do not lemmatize	(1)
‘pre_process’ function: stem tokens	Use PorterStemmer from nltk package	No improvement	Do not stem	(1)
‘train_classifier’ function: tunning regularizing cost hyperparameter	Use GridSearchCV method for tunning a list of C values	Greatly improve all score with C = 0.01	Update C = 0.01	Remove stop-words and set the model hyperparameter C = 0.01 (2)
‘train_classifier’ function: update balanced class_weight	Set a hyperparameter ‘class_weight’ as ‘balanced’	No improvement	Do not balance class weight	(2)
‘to_feature_vector’ function: add control for vocabulary	Set minimum frequency = 2	No improvement	Do not set control for vocabulary	(2)
‘to_feature_vector’ function: add sentence length	Add a token for length of text	No improvement	Do not add text length	(2)

Eventually, after testing around, we have optimal preprocessing method: ***(a)*** tokenize with str.split(), ***(b)*** remove stopwords and ***(c)*** set the C hyperparameter of LinearSVC as 0.01. The result achieved is significantly improved, compared to base model(3): precision/recall/f1 score/accuracy of the new one is 5%-7% better.

6. Question 6

The strategy is to add the feature one by one, then check the individual effect on the score, benchmarking with the best model achieved in question 5. Below is the summary:

Table 2: Individual feature testing (question 6)

Individual feature test	**Impact**	**Decision**
Subject	All the scores improve	Add ‘subject’ to the model
Speaker	All the scores improve	Add ‘speaker’ to the model
Speaker job title	All the scores improve	Add ‘speaker job title’ to the model
State info	All the scores improve	Add ‘state info’ to the model
Party affiliation	All the scores improve	Add ‘party affiliation’ to the model
Total barely true counts	All the scores improve	Add ‘total barely true counts’ to the model
Total false counts	All the scores improve	Add ‘total false counts’ to the model
Total half true counts	All the scores improve	Add ‘total half true counts’ to the model
Total mostly true counts	All the scores improve	Add ‘total mostly true counts’ to the model
Total pants on fire counts	All the scores improve	Add ‘total pants on fire’ to the model
Context	No improvement	Do not add ‘context’ to the model

After adding more features as stated in the table, the performance score of new model, with 10 new added features (from table 2), is improved by 21%-22% on test dataset prediction

(3) Base model: the model which is built throughout question 1 – question 4