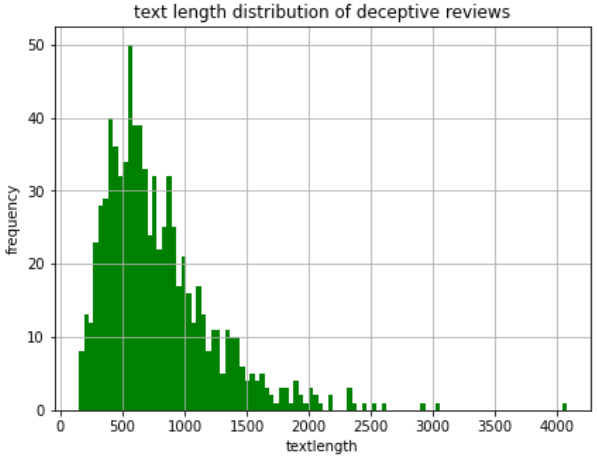
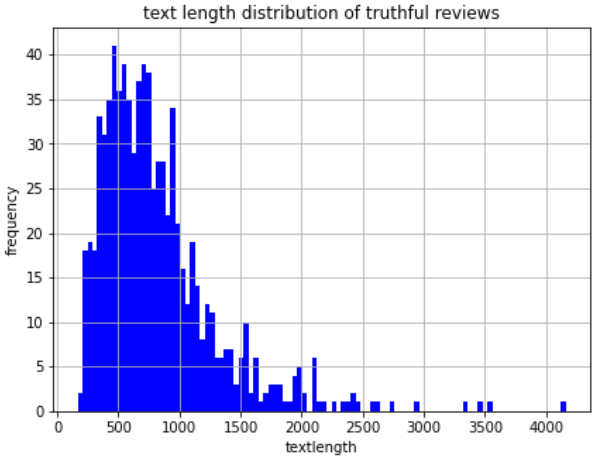
A. Data Analysis and Visualizations:

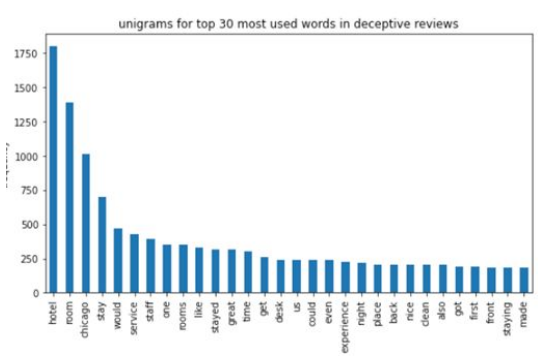
To visualize the data, we used the WordCloud API that provides a clear visualization of which words were most frequently used in the data based on the font size of the words among both truthful and deceptive reviews.

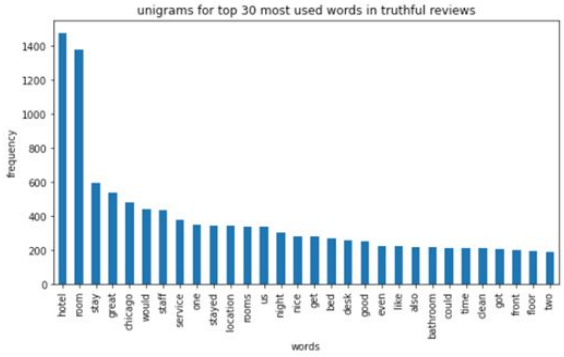
We noticed that irrelevant words such as “husband” and “wife” were more frequently used in deceptive reviews whereas truthful reviews frequently used more generalized words such as “location”, “chicago”, “bed”, “lobby”, etc. However, there were still numerous frequently used words that were shared among both truthful and deceptive reviews. Both types of reviews still frequently used words such as “great”, “bathroom”, and “staff”. Additionally, we also analyzed the frequency distributions of both truthful and deceptive reviews by measuring the proportion of reviews that were of specific text lengths. The graphs are shown down below:



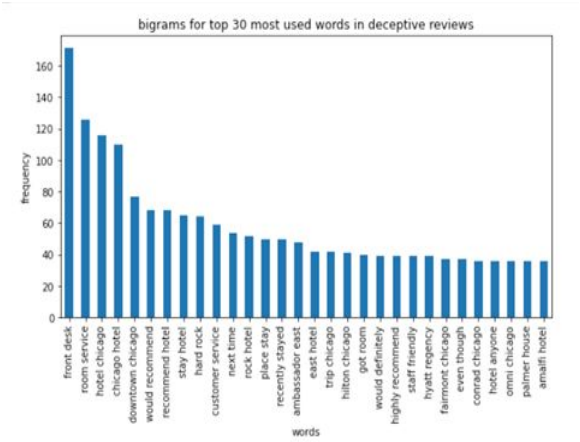


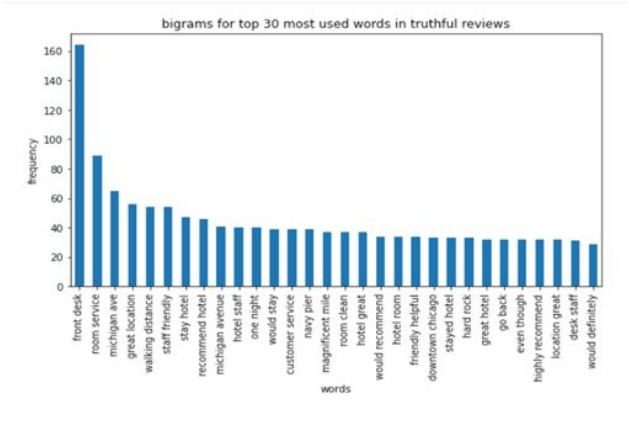
The text lengths for truthful reviews are more distributed than the text lengths found in deceptive reviews. The text lengths are more spread out in truthful reviews in comparison. However, the distributions share many similarities such as its right tailed distribution which shifts the mean for the reviews towards text lengths of roughly 500-1000 rather than higher text lengths. We also analyzed unigrams and bigrams for the top 30 most frequently used words in both deceptive and truthful reviews. The data visualization for unigrams is shown below:





As mentioned before, there were what seemed to be many frequently used words among both deceptive and truthful reviews that were more discovered using the WordCloud API. Upon further inspection by plotting the most frequently used unigrams, we can conclude that the frequently used word shared amongst both types of reviews was “hotel”, “room”, “chicago”, “stay”, and “front” whereas the rest of the words were completely different amongst both types of reviews. However, Chicago was used 1000 times in the deceptive reviews yet only used approximately 400 to 600 times in truthful reviews. A similar relationship can be found between the most frequently used bigrams as shown below:





There are even words that are shared amongst a few bigrams in both deceptive and truthful reviews as well. Both deceptive and truthful reviews share the bigram “would recommend”.

B.Data Preprocessing:

We performed data preprocessing using the nltk library in order to remove meaningless, noisy data and because working with a significantly larger dataset is more computationally intensive. As such, for the data preprocessing:

1. We converted all of the words in each review to lowercase letters, removed numbers, and removed punctuation.
2. We removed stop words to filter out common words that don’t signify any sort of relationship between the data and the class labels. These stop words include: you, he, she, who, do, is, above, that, in, will, can, etc. which were extracted using the nltk.corpus library which

provides a dictionary for all stop words present in the english library and filtered them out as we processed each line in the files.

3. We performed lemmatization on the data which is the process of group words together by removing their suffixes, so they can be analyzed as a single item. Contrary to stemming algorithms which also removes suffixes of each word, lemmatization removes suffixes based on the context of the word such as whether that word is being used as a noun, verb, adverb, and adjective. This is done through a pos tag which is processed along with the word in order to provide the base form of the word. This gives more precise results than stemming algorithms, but as a result, lemmatization is more computationally intensive. However, we selected lemmatization because accuracy was our main concern. Through multiple trials and errors, data preprocessing was handled relatively quickly which reinforced our original assumption: computational overhead was a miniscule cost to the performance benefits lemmatization provides. Lemmatization was extracted using the nltk.stem library which enables us to represent each line as a list of tuples where each tuple contains its word with its respective pos tag which denotes the context of the word.
4. After data cleaning was performed, we then mapped our data to vectors ranging from 0 to 1. Machine learning models can only process numerical data, so converting our data to a matrix of values that were relevant in spam detection was crucial in order to produce the best results. Because analyzing the frequencies of words were critical in our data analysis and visualization, we selected the tfidf vectorization which gives our machine learning model relevant information such as the frequency of the word as well as its relative occurrence over our corpus. As our training results conclude, selecting the tfidf matrix provided tremendous results.
5. Here is some examples after cleaning the data:

Review	Cleaned Review
excellent staff and customer service, very clean and spotless. elegant and luxurious with a beautiful ocean view. the bed is very comfortable and relaxing. i give it a five star.	['excellent', 'staff', 'customer', 'service', 'clean', 'spotless', 'elegant', 'luxurious', 'beautiful', 'ocean', 'view', 'bed', 'comfortable', 'relax', 'give', 'five', 'star']
My husband and I recently stayed at the Hard Rock Hotel Chicago and we can't wait to go back! The hotel is located in downtown Chicago and seems to be at the heart of the city, we were close to everything. The Hard Rock Hotel is forty stories high and the view from our room was simply breathtaking. The room itself was spotless and featured modern luxurious decor and furnishings. The bed was heavenly. The hotel staff was friendly, upbeat and extremely helpful, we asked for directions and restaurant recommendations and they were spot on with both. I highly recommend the Hard Rock Hotel in Chicago and I would stay there again in a heartbeat.	['husband', 'recently', 'stay', 'hard', 'rock', 'hotel', 'chicago', 'cant', 'wait', 'go', 'back', 'hotel', 'locate', 'downtown', 'chicago', 'seem', 'heart', 'city', 'close', 'everything', 'hard', 'rock', 'hotel', 'forty', 'story', 'high', 'view', 'room', 'simply', 'breathtaking', 'room', 'spotless', 'feature', 'modern', 'luxurious', 'decor', 'furnishing', 'bed', 'heavenly', 'hotel', 'staff', 'friendly', 'upbeat', 'extremely', 'helpful', 'ask', 'direction', 'restaurant', 'recommendation', 'spot', 'highly', 'recommend', 'hard', 'rock', 'hotel', 'chicago', 'would', 'stay', 'heartbeat']

C.Train-test split:

We all split the data in both 80% training and 20% testing data using the train_test_split method provided by the sklearn.model_selection package.

D.Experiment Setting:

First, we split the data in both training and testing data using the train_test_split method provided by the sklearn.model_selection package. We all split the training and testing data where our test size was 20% of the original data and shuffled the data to provide more relevant results in which our machine learning models weren't overly reliant on our train and test data. However, Nazim used a random seed of 42, Charchita used a random seed of 101, and Jing chose the default random seed that is used in the train_test_split method because these were seeds most commonly used when training machine learning models. While fine tuning our models, we concluded that selecting a seed wasn't as important as our accuracy scores were largely unchanged when adjusting the parameter. As such, the parameters that were selected were mostly done by convention.

Next, Charchita and Nazim used the Pipeline method provided by the sklearn.pipeline package which enabled us to combine multiple steps when creating our classifier. In our case, we combined the CountVectorizer and TfidfTransformer in order to map our data to the tfidf matrix. We both analyzed the data in respect to both unigrams and bigrams and adjusted the max_features that were observed accordingly. Charchita selected 4000 for the features being observed whereas Nazim selected 1000 for the training data and 1000 for the test data. Further analysis concludes that selecting the max_features above a threshold of 1000 for the train data produces the best results.

For the support vector machine model, there were 4 parameters to adjust, C which is the regularization parameter, a kernel function which is the mathematical function in which defines how reviews are being separated amongst different class labels, gamma which is the coefficient of the kernel function, and the degree of the polynomial function if it were used as the kernel function. GridSearchCV from the sklearn.model_selection package was used to provide an approximation for the best hyperparameters to select for the classifiers for both training and testing data. Hyperparameters were adjusted not only to provide the best performance results, but also to ensure that the data was not overfitted or underfitted. For instance, C and Gamma were adjusted along with the number of features so that not only were enough features used for training the model, but also to ensure that C and Gamma weren't too small or too large because those influence the number of data points being used for the support vectors. Furthermore, the multinomial Naive Bayes model was used for Naive Bayes because it works well for data that has discrete features such as text length in text classification and default parameters were used.

For the neural network, Jing also used the Pipeline method provided by the sklearn.pipeline package and combined the CountVectorizer and TfidfTransformer in order to map our data to the tfidf matrix. At first, Jing used RNN with word-embedding to train the model, and she ran it on Python. She changed the value of learning rate, the structure, the optimizer, the batch size, and the parameters in lstm. After fine tuning, she used the lstm structure on Pytorch, Binary CrossEntropyLoss as the loss function, Adam algorithm as the optimizer with batch size 256 and learning rate 001. She only got good performance on train data, while didn't get good result

on test data. So she decided to use tfidf matrix to process the data. She changed the value of learning rate, the structure, the optimizer, and the batch size. After fine tuning, she used fully connected layer and activation function, Binary CrossEntropyLoss as the loss function, SGD algorithm as the optimizer with batch size 256 and learning rate 001. This time she got good performance on both train data and test data.

E.Evaluation Metrics:

We evaluated our machine learning models by computing its accuracy results as well our models' training times. Training time was evaluated using the time package where the time was computed before and after fitting the data and the difference in those times were computed. Accuracy scores were measured using the accuracy_score method provided by the sklearn.metrics package. For SVM in particular, we had to be cautious of overfitting and underfitting, so the model was repeatedly tested by fine tuning the number of features as well as the regularization parameters to ensure that the results given were reliable.

F.Hardware Specification:

Charchita tested her model using Google Colab, so the hardware that she used to test her model was run by the GPU provided by Google's cloud servers. In contrast, Nazim and Jing used different software when implementing their code. Because Jing's machine learning model used neural network, she ran her code using her own GPU. However, because Nazim used a support vector machine which is a model that is significantly less demanding, his model was tested using his CPU.

G.Results and Comparison of our models:

These were the results for our machine learning models:

	Training Time(s)	**Accuracy (test)**	**Accuracy (train)**
Supported Vector Machine	0.618974	100%	100%
Naive Bayes	0.279487	90.315%	96.796875%
Neural Network	3-4	90.625%	100%

H.Findings and conclusion:

From this experiment, we can conclude that the Support Vector Machine was the best model to use for text classification for the data we were given and tfidfVectorization is a tremendous tool when mapping the data to numerical data because it represents features that are critical in text classification such as word frequency which also had a significant impact on our data analysis and data visualization. Support Vector Machines however, proved to have significantly inferior training times in comparison to the Naive Bayes model. The training time for SVM can be adjusted based on the number of features observed, but in order to prevent underfitting and overfitting, an appropriate number of features was selected to ensure greater accuracy at the

expense of time. However, SVM's training time is significantly better than using recurrent neural networks in which the training time was 3-4 sec/epoch. Based on our training results, our models in general gave greater results when our models were fitted for training data rather than testing data. When tweaking hyperparameters for our Support Vector Machine, we noticed that the opposite were true where it performed better on test data compared to our training data. Regardless, every model we used had great results when using tfidfVectorization. Overall, Support Vector Machine provided the greatest accuracy results, but the worst training time.

I.Individual Contributions:

nazim	pre-process the data(remove stopwords, numbers and do the lemmatization), TF-IDF result analysis. Make the ppt and presentation. Train the SVM model. Make the most part of the report(pdf).
charchita	Data visualizations, analysis of model selection(advantages and disadvantages about different methods). Make the ppt. Train the naïve bayes model. Help revise the report(pdf).
Jing	upload the file to python, word-embedding, pre-process the data(do the lowercase, remove punctuations, help nazim about lemmatization). Make the ppt. Train the neural network model. Help revise the report(pdf).