A. Data Preparation

1. Importing the Dataset

We did download the dataset from the url *`https://github.com/GU4243-ADS/spring2018-project1-ginnyqg/raw/master/data/spooky.csv`* and save it locally to use it later, we did also display the first *10* samples

```
import pandas as pd
df = pd.read_csv('./ spooky.csv')
df.head(10)
```

B. Text Cleaning

1.Handling Repeated Characters

We write a function which uses regular expressions to handle repeated characters like "*cooooool*" and convert them to the canonical form "*cool*".

```
import re
def remove_repeated_chars(text):
    return re.sub(r'(.)\1+', r'\1\1', text)

df['text_text'] = df['text'].apply(remove_repeated_chars)
```

2. Handling Homoglyphs (e.g., "$tupide" to "stupide")

Homoglyphs are characters that look similar but have different meanings so we write a function that has a dictionary for common homoglyphs and it replaces them in the text variable given

```
def replace_homoglyphs(text):
    homoglyphs= {"$": "s", "@": "a", "0": "o", "3": "e"}
    for char, raplacement in homoglyphs.items():
        text = text.replace(char, replacement)
    return text

df['text'] = df['text'].apply(replace_homoglyphs)
```

3. Handling Special Entries (URLs, Emails, HTML Tags)

We used a function that uses regular expressions to detect and remove urls, email addresses and html tags from the given text

```
def clean_special_entries(text):
    text = re.sub(r'http\S+|www\S+|https\S+', '[URL]', text, flags=re.MULTILINE)
    text = re.sub(r'\S+@\S+', '[EMAIL]', text)
    text = BeautifulSoup(text, 'html.parser').get_text()
    return text

df['text'] = df['text'].apply(clean_special_entries)
```

4. Lowercasing the Text

We convert the text to lowercase

```
df['text'] = df['text'].str.lower()
```

5. Removing Punctuation

We use a function that remove punctuation using string library

```
import string
def remove_punc(text):
    return text.translate(str.maketrans('', '', string.punctuation))

df['text'] = df['text'].apply(remove_punc)
```

6. Removing Stop Words

We' use nltk's stopword list to filter out common words that don’t add much value/meaning to our text, it is built-in list of common words like "the", "and", etc, we define that the used language is english

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
```

```
stop_words = set(stopwords.words('english'))
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
```

7. Detecting and Translating Languages

We use langdetect library for detecting the used language of our text then googletrans library to translate it into the most frequent language (english)

```
from langdetect import detect
from googletrans import Translator
translator = Translator()
def detect_and_translate(text):
    try:
        lang = detect(text)
        if lang != 'en':
            text = translator.translate(text, dest='en').text
    except:
        pass
    return text

df['text'] = df['text'].apply(detect_and_translate)
```

8. Removing Repeated Words

To remove repeated words from the text we create a function `remove_repeated_words` that will firstly split the given text into words using white space then for each word it will check if the current word is the same as the previous one

```
def remove_repeated_words(text):
    words = text.split()
    cleaned_words = [words[i] for i in range(len(words)) if i == 0 or words[i] != words[i-1]]
    return ' '.join(cleaned_words)


df['text'] = df['text'].apply(remove_repeated_words).str.strip()
```

C. Segmentation

1. Segmentation using white space/ponctuation

```
df['tokens_space'] = df['text'].apply(lambda x: x.split())
```

2. Segmentation using algorithm

```
from nltk.tokenize import word_tokenize

df['tokens_rule'] = df['text'].apply(word_tokenize)
```

3. Segmentation based on subwords

Subword tokenization is a technique particularly useful for handling the out of vocabulary words, it breaks the giver word into smaller units, we will use the *transformers* library to implement this algorithm, we use *AutoTokenizer.from_pretrained("bert-base-uncased")* to load the tokenizer associated with *bert-base-uncased* model, which uses the *wordPiece* algorithm

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
df['tokens_subword'] = df['text'].apply(tokenizer.tokenize)
```

D. Named Entity Recognition (NER)

1. Named entities recognition with spacy

This will get in which category does the word belong to (date, place. etc)

```
import spacy

nlp = spacy.load('en_core_web_sm')
```

df['entities'] = df['text'].apply(lambda x: [(ent.text, ent.label_) for ent in nlp(x).ents])

```

### 2. POS tags analysis

This will determine what role this word is trying to be in the context of the text (noun, adjective, verb. etc)

```

df['pos_tags'] = df['text'].apply(lambda x: [(token.text, token.pos_) for token in nlp(x)])

```

# E. Lemmatization and Stemming

### lemmatization and stemming with nltk

Lemmatization will reduceswords to their base form (e.g., "running" → "run") and stemming will reduce words to their root (e.g., "looked" → "look")

```

from nltk.stem import WordNetLemmatizer, PorterStemmer
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
df['lemmatized'] = df['text'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in x.split()]))
df['stemmed'] = df['text'].apply(lambda x: ' '.join([stemmer.stem(word) for word in x.split()]))
```

# F. Frequency Analysis

### 1.Occurrences of ‘great’ bu author

We count the number of times the words "great" appear in the text and group it by author, first we will iterate on each instance in the text column and count how many times we did find the word ‘great’ then we use the function *`groupby`* to group the result by author

```

df['great_count'] = df['text'].apply(lambda x: x.lower().count('great'))

great_by_author = df.groupby('author')['great_count'].sum()
```

## 2.Visualization with pywaffle

Create a waffle chart summarizing the occurrences of *`great` word* by author

```

from pywaffle import Waffle

plt.figure(FigureClass=Waffle, rows=5, columns=10, values=great_by_author.to_dict(), legend={'loc': 'upper left'})

plt.show()
```

## 4.The most used words by author using wordcloud

We generate three word cloud, one for each author, to display the most used words in the text

```
*for author in df['author'].unique():*
    *text = ' '.join(df[df['author'] == author]['text'])*
    *generate_wordclound(text, author)*
```

5.Most 100 used words positives/negatives

Using *SentimentIntensityAnalyze* from *nltk.sentiment.vader,* which is pretrained sentiment analysis tool, we create a function named *classify_words* that classifies words in the given text as either positive or negative based on their sentiment score, the *sia.polarity_scores(word)* returns a dictionary containing the sentiment scores for the word, the scores include *neg* for *negative sentiment score*, *neu* for *neutral*, *pos* for *positive* and *compound*, the words with *compound* greater than 0 are classified as positive, the rest is negative, then we generate a wordcloud for each positive/negative words

```

*sia = SentimentIntensityAnalyzer()*

```
def classify_words(text):
    words = text.split()
    positive_words = [word for word in words if sia.polarity_scores(word)['compound'] > 0]
    negative_words = [word for word in words if sia.polarity_scores(word)['compound'] < 0]
    return " ".join(positive_words), " ".join(negative_words)

for author in df['author'].unique():
    text = " ".join(df[df['author'] == author]['text'])
    positive_text, negative_text = classify_words(text)
    generate_wordcloud(positive_text, f"Top Positive Words by {author}")
    generate_wordcloud(negative_text, f"Top Negative Words by {author}")
```

6.Analyse de sentiment

To determine whether the author test is predominantly positive, negative or neutral we will calculate the sentiment scores for each instance then aggregate it by author to calculate the mean

```
df['sentiment'] = df['text'].apply(lambda x: sia.polarity_scores(x)['compound'])

sentiment_counts = df.groupby('author')['sentiment'].mean()
```

7.Best methode to use with this dataset

The methodes that we used(remove stopwords, urls, translate. etc), we choose them because our dataset contains the stopwords, urls and so, we used:

homoglyth & special chars handling to avoid analysis errors due to writing variations

removing stopwords to enhance the effectiveness of word frequency analysis by focusing on meaningful content words

lemmatization and stemming to retain the original meaning of the words

language detecting & translation to ensure all the text is processed in one defined language

NER & POS tagging named entity recognition helps in identifying key subjects such as names, places, organizations, etc, and pos tagging to understand the grammatical structure of the text

wordcloud & pywaffle to summarize key word patterns, and make it easier to identify trends in the text

sentiment analysis to capture the emotional tendencies of each author, this can be useful in determining writing styles and perspectives