Preprocessing:

1. Standard Preprocessing:

- Lowercasing: .lower() function to lower all text
- Remove Punctuations using string module:

```
tokens = nltk.word_tokenize(text)
filtered_tokens = [token for token in tokens if token not in
string.punctuation]
```

- Remove extra whitespaces:

```
text = text.strip()
```

- Removing Stop Words: (common words like a, an the etc that do not add significance)

```
# get list of stopwords in English
stopwords = nltk.corpus.stopwords.words("english")

# remove stopwords
filtered_tokens = [token for token in tokens if token.lower() not in stopwords]
```

- Remove HTML codes: We can use a regex expression pattern to remove such expressions.

```
pattern = r"<[^>]+>"

# replace HTML tags with an empty string
cleaned_text = re.sub(pattern, "", text)
```

- Remove Frequent words and Spelling corrections:

  For frequent words store words in a list and remove duplicates

  For spell checker use nltk library to get all words and replace them with correct words

```
# get list of English words
words = nltk.corpus.words.words()

# correct spelling of each word
corrected_tokens = []
for token in tokens:
  # find the word with the lowest edit distance
  corrected_token = min(words, key=lambda x: nltk.edit_distance(x, token))
                  corrected_tokens.append(corrected_token)
```

- Stemming and Lemmatization: Reduce all words to their basic forms ex running ->run etc
  For stemming:

  ```
  stemmer = nltk.stem.PorterStemmer()

  # stem each token
  stemmed_tokens = [stemmer.stem(token) for token in tokens]
  ```

  For Lemmatizing:

  ```
  lemmatizer = nltk.stem.WordNetLemmatizer()

  # lemmatize each token
                  lemmatized_tokens = [lemmatizer.lemmatize(token) for token in
                  tokens]
  ```

- Named entity recognition: We can use libraries like spacy or nltk to extract names and preprocess data

  ```
  tagged_tokens = nltk.pos_tag(tokens)

  # identify named entities
  named_entities = nltk.ne_chunk(tagged_tokens)
  ```

- Part of Speech tagging: Identifying the part of speech of each word in the text, such as noun, verb, or adjective.

  ```
  tokens = nltk.word_tokenize(text)

  # tag the tokens with their POS tags
  tagged_tokens = nltk.pos_tag(tokens)
  ```

2. Using ELK Stack like kibana ,Elastic Seacrh and Logstash:

We can use these tech stacks for storing, searching, analyzing, and visualizing textual data. Using elastic search we can store and index the data and perform search easily using various methods. We can also use it to tokenize filter and custom analyze the dataset. Kibana can be used to visualize this and help preprocess the data with searching and real time monitoring.

Data Workflow:

For ELK stack:

- Data Ingestion :-Use tools like Logstash or Beats to collect text data from various sources (e.g., social media, news articles, customer reviews) and ingest it into Elasticsearch.
- Text Processing :- Apply NLP preprocessing steps (e.g., tokenization, stop word removal, stemming) using Elasticsearch's built-in analyzers or custom analyzers.
- Indexing: Store and index the processed text data in Elasticsearch, making it ready for search and analysis.
- Search and Analysis: Perform full-text searches and complex queries to retrieve relevant text data. Use Elasticsearch aggregations to analyze term frequencies, co-occurrences, and other statistical properties of the text data.
- Visualization and Exploration:Create visualizations in Kibana to represent the analyzed data. Build interactive dashboards to explore the data and gain insights. Monitor real-time text data streams and set up alerts for significant changes.
- Reporting and Sharing: Generate and share reports based on the visualizations and dashboards in Kibana.