Journal Assignment

a. Key Insights:

Lab today was all about NLP preprocessing in Python. Got NLTK and spaCy set up, plus downloaded necessary data like punkt, stopwords, wordnet, and the POS tagger from NLTK, and the en_core_web_sm model for spaCy. Worked with different types of texts - simple, academic, social media, news, reviews - to see how preprocessing changes things. Did tokenization using NLTK's word_tokenize and spaCy's nlp. Also tackled stop word removal, comparing NLTK and spaCy's lists. Then looked at lemmatization and stemming, using PorterStemmer from NLTK and spaCy's lemmatizer. Finally, did text cleaning: lowercasing, removing extra spaces, punctuation, numbers, URLs, emails, mentions, and emojis. Built a preprocessing pipeline with minimal, standard, and aggressive settings, testing it on a product review and other samples, and checked how much it cut down on the number of tokens each time.

B. Things I Learned:

Today's lab really clicked for me with NLP preprocessing. I now see it's super important for cleaning up text so machines can actually understand it. It basically standardizes everything, gets rid of the junk, makes things less complicated, and helps the algorithms work better.

My experience with **version control using GitHub** reinforced its role in managing code versions and collaborating. The hypothetical setup of a GitHub repository prior to the lab would emphasize the importance of tracking changes and maintaining a project's history. Working with **Jupyter Notebook (Colab)** was highly beneficial due to its interactive nature. The ability to execute code cell by cell and see immediate outputs facilitated a hands-on learning experience for each preprocessing step. This interactive environment is ideal for experimentation and understanding the direct impact of each technique.
I specifically learned:

- **Tokenization Differences:** NLTK and spaCy handle contractions and punctuation differently during tokenization. While NLTK splits "It's" into 'It' and "'s'", and separates punctuation, spaCy often integrates punctuation more contextually, or separates it based on its linguistic model. I noted that spaCy seems to have a better grasp of parts of speech (POS) and linguistic depth, which could be more beneficial for tasks like information extraction.
- **Stop Word Nuances:** NLTK and spaCy have different stop word lists, which can significantly impact NLP results, especially in tasks like keyword extraction. spaCy's list is more extensive, leading to a greater vocabulary reduction when applied. I also learned that removing stop words is not always ideal, particularly in sentiment analysis where words like "not" are crucial for meaning.
- **Stemming vs. Lemmatization:** Lemmatization (spaCy) reduces words to their valid dictionary form considering context, providing higher accuracy, while stemming (NLTK's

PorterStemmer) is faster but might result in non-words by simply removing suffixes. For example, "flies" stemmed to "fli" and "was" to "wa", which are questionable, whereas lemmatization transformed "was" to "be" and "better" to "well", preserving meaning. I realized that lemmatization is generally more suitable for tasks requiring high accuracy and meaning preservation, like sentiment analysis or real-time chatbots, while stemming might be sufficient for search engines where speed and matching word variations are prioritized.

- **Text Cleaning Strategies:** Basic cleaning involves lowercase conversion, whitespace, punctuation, and number removal. Advanced cleaning tackles specific elements like URLs, emails, mentions, hashtags, and emojis, which are common in social media text. I learned that removing elements like emojis and hashtags can lead to information loss, potentially hurting NLP applications where emphasis or specific context (e.g., sentiment conveyed by an emoji) is important.

Challenges Faced and Overcoming Them:

Honestly, figuring out NLTK versus spaCy was a bit of a headache at first. I thought one was just plain better, but after looking at them side-by-side and asking some key questions, I got that it really depends on what you're trying to do. Just had to dive into the code and see what each one was spitting out and what got left behind. Plus, learned the hard way that sometimes taking out punctuation or common words can mess things up, which I only figured out by running into a few examples and asking more questions.