CISB5123 Text Analytics
Lab 4
Basic Text Pre-Processing

Text pre-processing is a process to clean and prepare the textual data before they can be used as the input for text modelling techniques. In this lab, we will apply text pre-processing techniques step-by-step on a dataset and store the result.

Step 1: Read the source data

```
# Load dataset
import pandas as pd

file_path = "Review.csv"
df = pd.read_csv(file_path)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df)
```

Step 2: Perform Text Pre-Processing

a. Convert text to lowercase

```
# Lowercase conversion
def convert_to_lowercase(text):
    return text.lower()

df["lowercased"] = df["Review"].apply(convert_to_lowercase)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["lowercased"])
```

b. Remove URLs

```
# Removal of URLs
import re

# remove any URLs that start with "http" or "www" from the text
def remove_urls(text):
    return re.sub(r'http\S+|www\S+', '', text)

df["urls_removed"] = df["lowercased"].apply(remove_urls)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["urls_removed"])
```

c. Remove HTML tags

```
# Removal of HTML tags
from bs4 import BeautifulSoup

# extracts only the text, removing all HTML tags
def remove_html_tags(text):
    return BeautifulSoup(text, "html.parser").get_text()
```

```
df["html_removed"] = df["urls_removed"].apply(remove_html_tags)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["html_removed"])
```

d. Remove emojis

```
# Removal of emojis (if any)
import emoji

# replace emoji with ''
def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

df["emojis_removed"] = df["html_removed"].apply(remove_emojis)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["emojis_removed"])
```

e. Replace internet slang/chat words

```
# Replace internet slang/chat words
# Dictionary of slang words and their replacements
slang_dict = {
    "tbh": "to be honest",
    "omg": "oh my god",
    "lol": "laugh out loud",
    "idk": "I don't know",
    "brb": "be right back",
    "btw": "by the way",
    "imo": "in my opinion",
    "smh": "shaking my head",
    "fyi": "for your information",
    "np": "no problem",
```

```
    "ikr": "I know right",
    "asap": "as soon as possible",
    "bff": "best friend forever",
    "gg": "good game",
    "hmu": "hit me up",
    "rofl": "rolling on the floor laughing"
}

# Function to replace slang words
def replace_slang(text):
    # Create a list of escaped slang words
    escaped_slang_words = []  # Empty list to store escaped slang words

    for word in slang_dict.keys():
        escaped_word = re.escape(word)  # Ensure special characters are escaped
        escaped_slang_words.append(escaped_word)  # Add to list

    # Join the words using '|'
    slang_pattern = r'\b(' + '|'.join(escaped_slang_words) + r')\b'

    # Define a replacement function
    def replace_match(match):
        slang_word = match.group(0)  # Extract matched slang word
        return slang_dict[slang_word.lower()]  # Replace with full form

    # Use regex to replace slang words with full forms
    replaced_text = re.sub(slang_pattern, replace_match, text, flags=re.IGNORECASE)

    return replaced_text


# Apply the function to the column
df["slangs_replaced"] = df["emojis_removed"].apply(replace_slang)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["slangs_replaced"])
```

f. Replace contractions

```
# Replace Contractions
contractions_dict = {
    "wasn't": "was not",
    "isn't": "is not",
    "aren't": "are not",
    "weren't": "were not",
    "doesn't": "does not",
    "don't": "do not",
    "didn't": "did not",
    "can't": "cannot",
    "couldn't": "could not",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "won't": "will not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "i'm": "i am",
    "you're": "you are",
    "he's": "he is",
    "she's": "she is",
    "it's": "it is",
    "we're": "we are",
    "they're": "they are",
    "i've": "i have",
    "you've": "you have",
    "we've": "we have",
    "they've": "they have",
    "i'd": "i would",
    "you'd": "you would",
    "he'd": "he would",
    "she'd": "she would",
    "we'd": "we would",
    "they'd": "they would",
    "i'll": "i will",
```

```
    "you'll": "you will",
    "he'll": "he will",
    "she'll": "she will",
    "we'll": "we will",
    "they'll": "they will",
    "let's": "let us",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "when's": "when is",
    "why's": "why is"
}

# Build the regex pattern for contractions
escaped_contractions = []  # List to store escaped contractions

for contraction in contractions_dict.keys():
    escaped_contraction = re.escape(contraction)  # Escape special characters (e.g., apostrophes)
    escaped_contractions.append(escaped_contraction)  # Add to list

# Join the escaped contractions with '|'
joined_contractions = "|".join(escaped_contractions)

# Create a regex pattern with word boundaries (\b)
contractions_pattern = r'\b(' + joined_contractions + r')\b'

# Compile the regex
compiled_pattern = re.compile(contractions_pattern, flags=re.IGNORECASE)

# Define a function to replace contractions
def replace_contractions(text):
    # Function to handle each match found
    def replace_match(match):
        matched_word = match.group(0)  # Extract matched contraction
        lower_matched_word = matched_word.lower()  # Convert to lowercase
```

```
        expanded_form = contractions_dict[lower_matched_word]  # Get full form from dictionary
        return expanded_form  # Return the expanded form

    # Apply regex substitution
    expanded_text = compiled_pattern.sub(replace_match, text)

    return expanded_text  # Return modified text

# Apply the function to a DataFrame column
df["contractions_replaced"] = df["slangs_replaced"].apply(replace_contractions)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["contractions_replaced"])
```

g. Remove punctuations and special characters

```
# Remove punctuations and special characters
import string

# Function to remove punctuation
def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

# Apply the function to the column
df["punctuations_removed"] = df["contractions_replaced"].apply(remove_punctuation)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["punctuations_removed"])
```

h. Remove numbers

```
# Remove numbers
def remove_numbers(text):
    return re.sub(r'\d+', '', text)  # Removes all numeric characters

# Apply the function to the column
df["numbers_removed"] = df["punctuations_removed"].apply(remove_numbers)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["numbers_removed"])
```

i. Correct spelling mistakes

```
# Correct spelling mistakes
from autocorrect import Speller

# Initialize spell checker
spell = Speller(lang='en')

# Function to correct spelling
def correct_spelling(text):
    return spell(text)  # Apply correction

# Apply the function to the column
df["spelling_corrected"] = df["numbers_removed"].apply(correct_spelling)
```

```
# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["spelling_corrected"])
```

j. Remove stopwords

```
# Remove stopwords
import nltk
from nltk.corpus import stopwords

# Download stopwords if not already downloaded
nltk.download('stopwords')

# Define stopwords list
stop_words = set(stopwords.words('english'))

# Function to remove stopwords
def remove_stopwords(text):
    words = text.split()  # Split text into words
    filtered_words = []  # Create an empty list to store words after stopword removal

    for word in words:  # Loop through each word in the list of words
        lower_word = word.lower()  # Convert the word to lowercase for uniform
comparison
```

```
        if lower_word not in stop_words:  # Check if the lowercase word is NOT in the stopwords list
            filtered_words.append(word)  # If it's not a stopword, add it to the filtered list

    return " ".join(filtered_words)  # Join words back into a sentence

# Apply the function to the column
df["stopwords_removed"] = df["spelling_corrected"].apply(remove_stopwords)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["stopwords_removed"])
```

k. Stemming - - reduces words to their base root by chopping off suffixes

```
# Stemming - reduces words to their base root by chopping off suffixes
from nltk.stem import PorterStemmer

# Initialize the stemmer
stemmer = PorterStemmer()

# Function to apply stemming
def stem_text(text):
    if not isinstance(text, str):
        return ""
```

```
    words = text.split()
    stemmed_words = [stemmer.stem(word) for word in words]  # Apply stemming
    return " ".join(stemmed_words)

# Apply the function
df["stemmed_words"] = df["stopwords_removed"].apply(stem_text)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["stemmed_words"])
```

I. Lemmatization - reduces words to their base dictionary form (lemma)

```
import nltk

# Download the required resources
nltk.download('wordnet')                    # For lemmatization
nltk.download('omw-1.4')                    # WordNet lexical database
nltk.download('averaged_perceptron_tagger_eng')  # For POS tagging
nltk.download('punkt_tab')                  # For tokenization

# Lemmatization - reduces words to their base dictionary form (lemma)
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk import pos_tag
```

```
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to map NLTK POS tags to WordNet POS tags
def get_wordnet_pos(nltk_tag):
    if nltk_tag.startswith('J'):  # Adjective
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):  # Verb
        return wordnet.VERB
    elif nltk_tag.startswith('N'):  # Noun
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):  # Adverb
        return wordnet.ADV
    else:
        return wordnet.NOUN  # Default to noun

# Function to lemmatize text with POS tagging
def lemmatize_text(text):
    if not isinstance(text, str):  # Ensure input is a string
        return ""

    words = word_tokenize(text)  # Tokenize text into words
    pos_tags = pos_tag(words)  # Get POS tags

    # Lemmatize each word with its correct POS tag
```

```
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(tag)) for word, tag in pos_tags]

    return " ".join(lemmatized_words)  # Join words back into a sentence

# Apply the function to the column
df["lemmatized"] = df["stopwords_removed"].apply(lemmatize_text)

# Display column content without truncation
pd.set_option('display.max_colwidth', None)  # Set to None for unlimited width
print(df["lemmatized"])
```

Step 3: Save the result to a file

```
df.to_csv("Processed_Reviews.csv", index=False)  # Saves without the index column
```

Putting it all together

```
import pandas as pd
import re
import emoji
import string
import nltk

from bs4 import BeautifulSoup
from autocorrect import Speller
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')                    # For lemmatization
nltk.download('omw-1.4')                    # WordNet lexical database
nltk.download('averaged_perceptron_tagger_eng')  # For POS tagging
nltk.download('punkt_tab')                  # For tokenization

# Initialize tools
spell = Speller(lang='en')
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Dictionary of slang words and their replacements
slang_dict = {
    "tbh": "to be honest",
    "omg": "oh my god",
    "lol": "laugh out loud",
    "idk": "I don't know",
    "brb": "be right back",
    "btw": "by the way",
    "imo": "in my opinion",
    "smh": "shaking my head",
    "fyi": "for your information",
    "np": "no problem",
    "ikr": "I know right",
    "asap": "as soon as possible",
```

```
    "bff": "best friend forever",
    "gg": "good game",
    "hmu": "hit me up",
    "rofl": "rolling on the floor laughing"
}

# Contractions dictionary
contractions_dict = {
    "wasn't": "was not",
    "isn't": "is not",
    "aren't": "are not",
    "weren't": "were not",
    "doesn't": "does not",
    "don't": "do not",
    "didn't": "did not",
    "can't": "cannot",
    "couldn't": "could not",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "won't": "will not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "i'm": "i am",
    "you're": "you are",
    "he's": "he is",
    "she's": "she is",
    "it's": "it is",
    "we're": "we are",
    "they're": "they are",
    "i've": "i have",
    "you've": "you have",
    "we've": "we have",
    "they've": "they have",
    "i'd": "i would",
    "you'd": "you would",
    "he'd": "he would",
    "she'd": "she would",
    "we'd": "we would",
    "they'd": "they would",
```

```
    "i'll": "i will",
    "you'll": "you will",
    "he'll": "he will",
    "she'll": "she will",
    "we'll": "we will",
    "they'll": "they will",
    "let's": "let us",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "when's": "when is",
    "why's": "why is"
}

# Remove any URLs that start with "http" or "www" from the text
def remove_urls(text):
    return re.sub(r'http\S+|www\S+', '', text)

# extracts only the text, removing all HTML tags
def remove_html(text):
    return BeautifulSoup(text, "html.parser").get_text()

# replace emoji with ''
def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

# Replace internet slang/chat words
def replace_slang(text):
    # Create a list of escaped slang words
    escaped_slang_words = []  # Empty list to store escaped slang words

    for word in slang_dict.keys():
        escaped_word = re.escape(word)  # Ensure special characters are escaped
        escaped_slang_words.append(escaped_word)  # Add to list

    # Join the words using '|'
    slang_pattern = r'\b(' + '|'.join(escaped_slang_words) + r')\b'

    # Define a replacement function
```

```
    def replace_match(match):
        slang_word = match.group(0)  # Extract matched slang word
        return slang_dict[slang_word.lower()]  # Replace with full form

    # Use regex to replace slang words with full forms
    replaced_text = re.sub(slang_pattern, replace_match, text, flags=re.IGNORECASE)

    return replaced_text

# Function to expand contractions
# Build the regex pattern for contractions
escaped_contractions = []  # List to store escaped contractions

for contraction in contractions_dict.keys():
    escaped_contraction = re.escape(contraction)  # Escape special characters (e.g.,
apostrophes)
    escaped_contractions.append(escaped_contraction)  # Add to list

# Join the escaped contractions with '|'
joined_contractions = "|".join(escaped_contractions)

# Create a regex pattern with word boundaries (\b)
contractions_pattern = r'\b(' + joined_contractions + r')\b'

# Compile the regex
compiled_pattern = re.compile(contractions_pattern, flags=re.IGNORECASE)

# Define a function to replace contractions
def replace_contractions(text):
    # Function to handle each match found
    def replace_match(match):
        matched_word = match.group(0)  # Extract matched contraction
        lower_matched_word = matched_word.lower()  # Convert to lowercase
        expanded_form = contractions_dict[lower_matched_word]  # Get full form from
dictionary
        return expanded_form  # Return the expanded form

    # Apply regex substitution
    expanded_text = compiled_pattern.sub(replace_match, text)
```

```
    return expanded_text  # Return modified text

# Function to remove punctuation
def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

# Function to remove numbers
def remove_numbers(text):
    return re.sub(r'\d+', '', text)

# Function to correct spelling using AutoCorrect
def correct_spelling(text):
    return spell(text)  # Apply correction

# Function to remove stopwords
def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return " ".join(filtered_words)

# Function to map NLTK POS tags to WordNet POS tags
def get_wordnet_pos(nltk_tag):
    if nltk_tag.startswith('J'):  # Adjective
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):  # Verb
        return wordnet.VERB
    elif nltk_tag.startswith('N'):  # Noun
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):  # Adverb
        return wordnet.ADV
    else:
        return wordnet.NOUN  # Default to noun

# Function to lemmatize text with POS tagging
def lemmatize_text(text):
    if not isinstance(text, str):  # Ensure input is a string
        return ""

    words = word_tokenize(text)  # Tokenize text into words
    pos_tags = pos_tag(words)  # Get POS tags
```

```
    # Lemmatize each word with its correct POS tag
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(tag)) for
word, tag in pos_tags]

    return " ".join(lemmatized_words)  # Join words back into a sentence

# Function to tokenize text
def tokenize_text(text):
    if not isinstance(text, str):  # Ensure the input is a string
        return []
    return word_tokenize(text)  # Tokenize text into words

# Function to apply all preprocessing steps
def preprocess_text(text):
    text = text.lower()          # Step 1: Lowercasing
    text = remove_urls(text)             # Step 2: Remove URLs
    text = remove_html(text)              # Step 3: Remove HTML tags
    text = remove_emojis(text)            # Step 4: Remove Emojis
    text = replace_slang(text)           # Step 5: Replace Slang
    text = replace_contractions(text)       # Step 6: Expand Contractions
    text = remove_punctuation(text)       # Step 7: Remove Punctuation
    text = remove_numbers(text)           # Step 8: Remove Numbers
    text = correct_spelling(text)         # Step 9: Correct Spelling
    text = remove_stopwords(text)         # Step 10: Remove Stopwords
    text = lemmatize_text(text)           # Step 11: Lemmatization
    text = tokenize_text(text)           # Step 12: Tokenization
    return text

# Load dataset
df = pd.read_csv("Review.csv")  # Replace with your file

# Apply preprocessing pipeline
df["processed"] = df["Review"].apply(preprocess_text)

# Save the cleaned dataset
df.to_csv("Processed_Reviews2.csv", index=False)

# Display the first few rows
print(df[["Review", "processed"]].head())
```

Exercise

1. Identify the issues with the "Review" column in the UNITENReview.csv file
2. Perform the necessary text pre-processing steps based on the identified issues
3. Save the result in a .csv file