PREPROCESSING CODE

```
import re
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Load data
data =
pd.read_csv('/Users/pillisachethan/Downloads/combined_data.csv')
data['post'].fillna('', inplace=True)

# Convert to lowercase
data['lowercase'] = data['post'].str.lower()

# Remove special characters
data['no_special_chars'] = data['lowercase'].apply(
    lambda x: re.sub(r'[^a-z\s]', '', x) if isinstance(x, str) else x
)
print(data[['lowercase', 'no_special_chars']].head())
```

```
# Set up stop words
nltk.download('stopwords')
nltk.download('punkt')
stop_words = set(stopwords.words('english'))

# Function to remove stop words
def remove_stopwords(text):
    tokens = word_tokenize(text)  # Tokenize the text
    return ' '.join([word for word in tokens if word not in stop_words])

# Apply stop words removal
data['no_stopwords'] =
data['no_special_chars'].apply(remove_stopwords)
print(data[['no_special_chars', 'no_stopwords']].head())

# Tokenize text
data['tokens'] = data['no_stopwords'].apply(word_tokenize)
print(data[['no_stopwords', 'tokens']].head())

# Custom Porter Stemmer Implementation
class porter:
    def isvowel(self, l):
        return l.lower() in 'aeiou'
```

```
def iscons(self, l):
    return not self.isvowel(l)

def form(self, word):
    return ''.join(['V' if self.isvowel(c) else 'C' for c in word])

def m_count(self, word):
    form_string = self.form(word)
    return form_string.count('VC')

def get_base(self, word, suf):
    suflen = word.rfind(suf)
    return word[:suflen] if suflen != -1 else word

def replacer(self, word, suf1, suf2):
    return self.get_base(word, suf1) + suf2

def contains_vowel(self, word):
    return any(self.isvowel(c) for c in word)

def CC(self, word):
    return self.form(word)[-2:] == 'CC'

def CVC(self, word):
```

```
        return self.form(word)[-3:] == 'CVC' and word[-1] not in 'wxyz'

    def step_1a(self, word):
        if word.endswith('sses'):
            return self.replacer(word, 'sses', 'ss')
        elif word.endswith('ies'):
            return self.replacer(word, 'ies', 'i')
        elif word.endswith('ss'):
            return word
        elif word.endswith('s'):
            return self.replacer(word, 's', '')
        return word

    def step_1b(self, word):
        if word.endswith('eed'):
            if self.m_count(self.get_base(word, 'eed')) > 0:
                return self.replacer(word, 'eed', 'ee')
        elif word.endswith('ed') and
self.contains_vowel(self.get_base(word, 'ed')):
            word = self.get_base(word, 'ed')
            return self.part_1b(word)
        elif word.endswith('ing') and
self.contains_vowel(self.get_base(word, 'ing')):
            word = self.get_base(word, 'ing')
```

```
            return self.part_1b(word)
        return word

    def part_1b(self, word):
        if word.endswith('at') or word.endswith('bl'):
            return word + 'e'
        elif self.CC(word) and word[-1] not in 'lsz':
            return word[:-1]
        elif self.m_count(word) == 1 and self.CVC(word):
            return word + 'e'
        return word

    def step_1c(self, word):
        if self.contains_vowel(word) and word.endswith('y'):
            return self.replacer(word, 'y', 'i')
        return word

    def step_2(self, word):
        if word.endswith('ational'):
            return self.replacer(word, 'ational', 'ate')
        elif word.endswith('ization'):
            return self.replacer(word, 'ization', 'ize')
        elif word.endswith('biliti'):
            return self.replacer(word, 'biliti', 'ble')
```

```
        return word

    def step_3(self, word):
        if word.endswith('icate'):
            return self.replacer(word, 'icate', 'ic')
        elif word.endswith('ful'):
            return self.replacer(word, 'ful', '')
        elif word.endswith('ness'):
            return self.replacer(word, 'ness', '')
        return word

    def step_4(self, word):
        suffixes = ['ance', 'ent', 'ive']
        for suffix in suffixes:
            if word.endswith(suffix) and self.m_count(self.get_base(word,
suffix)) > 1:
                return self.replacer(word, suffix, '')
        return word

    def step_5a(self, word):
        if word.endswith('e') and self.m_count(self.get_base(word, 'e')) >
1:
            return self.replacer(word, 'e', '')
        return word
```

```
    def step_5b(self, word):
        if self.CC(word) and word.endswith('l') and self.m_count(word) >
1:
            return word[:-1]
        return word

# Instantiate custom stemmer
custom_stemmer = porter()

# Function to stem words
def stem_words(tokens):
    stemmed_words = []
    for word in tokens:
        stem_word = custom_stemmer.step_1a(word)
        stem_word = custom_stemmer.step_1b(stem_word)
        stem_word = custom_stemmer.step_1c(stem_word)
        stem_word = custom_stemmer.step_2(stem_word)
        stem_word = custom_stemmer.step_3(stem_word)
        stem_word = custom_stemmer.step_4(stem_word)
        stem_word = custom_stemmer.step_5a(stem_word)
        stem_word = custom_stemmer.step_5b(stem_word)
        stemmed_words.append(stem_word)
    return stemmed_words
```

```
# Apply stemming to tokens
data['stemmed'] = data['tokens'].apply(stem_words)
print(data[['tokens', 'stemmed']].head())
```