**#Python script for text preprocessing**

%pip install pySpellChecker

%pip install emoji

%pip install --upgrade textblob

%pip install langdetect

import pandas as pd

import numpy as np

import nltk

import string

import re

import emoji

import spacy

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

from spellchecker import SpellChecker

from textblob import TextBlob

from langdetect import detect

nltk.download('punkt')

nltk.download('wordnet')

datapath = "/content/combined data.xlsx"

df = pd.read\_excel(datapath)

df["translated\_text"] = df["translated\_text"].astype(str)

contractions\_mapping = {

"ain't": "is not",

"aren't": "are not",

"can't": "cannot",

"'cause": "because",

"could've": "could have",

"couldn't": "could not",

"didn't": "did not",

"doesn't": "does not",

"don't": "do not",

"hadn't": "had not",

"hasn't": "has not",

"haven't": "have not",

"he'd": "he would",

"he'll": "he will",

"he's": "he is",

"how'd": "how did",

"how'll": "how will",

"how's": "how is",

"i'd": "I would",

"i'll": "I will",

"i'm": "I am",

"i've": "I have",

"isn't": "is not",

"it'd": "it would",

"it'll": "it will",

"it's": "it is",

"let's": "let us",

"ma'am": "madam",

"might've": "might have",

"mightn't": "might not",

"must've": "must have",

"mustn't": "must not",

"needn't": "need not",

"shan't": "shall not",

"she'd": "she would",

"she'll": "she will",

"she's": "she is",

"should've": "should have",

"shouldn't": "should not",

"that's": "that is",

"there's": "there is",

"they'd": "they would",

"they'll": "they will",

"they're": "they are",

"they've": "they have",

"wasn't": "was not",

"we'd": "we would",

"we'll": "we will",

"we're": "we are",

"we've": "we have",

"weren't": "were not",

"what'll": "what will",

"what're": "what are",

"what's": "what is",

"what've": "what have",

"when's": "when is",

"where'd": "where did",

"where's": "where is",

"who'll": "who will",

"who's": "who is",

"won't": "will not",

"would've": "would have",

"wouldn't": "would not",

"you'd": "you would",

"you'll": "you will",

"you're": "you are",

"you've": "you have"

}

# Function to expand contractions

def expand\_contractions(text, contractions\_mapping):

contractions\_pattern = re.compile('({})'.format('|'.join(contractions\_mapping.keys())),

flags=re.IGNORECASE | re.DOTALL)

def expand\_match(contraction):

match = contraction.group(0)

first\_char = match[0]

expanded\_contraction = contractions\_mapping.get(match.lower())

expanded\_contraction = first\_char + expanded\_contraction[1:]

return expanded\_contraction

expanded\_text = contractions\_pattern.sub(expand\_match, text)

return expanded\_text

df["expanded\_text"] = df["Text"].apply(lambda x: expand\_contractions(x, contractions\_mapping) if isinstance(x, str) else x)

# Display DataFrame

print(df)

nlp = spacy.load("en\_core\_web\_sm")

def preprocessor(text):

print(f"Input text: {text}")

if text is None or pd.isnull(text): # Check for None or NaN values

return ""

#spell\_checker = SpellChecker(language='en')

spell\_checker = SpellChecker(language='en') if text.lower() != "hammanskraal" else None

clean\_tokens = []

stop\_words = {'their','anyone', 'angithi', 'me', 'under', 'be', 'they', 'he', 'other', 'yourselves', 're', 'at', 'your', 'between', 'been', 'you', 'on', 'then', 'most', 'ourselves', 'all', 'out', 's', 'does', 'has', 'myself', 'of', 'yours', "it's", 'by', "that'll", 'for', 'from', 'yourself', 'do', 'whom', 'because', 'i', 'she', 'once', 'but', 'we', 'these', 'where','doing', 'what', 'can', 'should', 'her', 'ours', 'few', 'or', 'again', 'than', 'that', 'there', 'are', 'each', 'am', 'here', 'just', 'now', 'so', 'being', 'in', 'same', 'theirs', 'them', 'those', 'his', 'such', 'over', 'a', 'have', 'hers', "you'd", 'our', 'further', 'themselves', 'through', 'very', 'herself', "you've", 'below', 'did', 'if', 'will', 'why', 'its', 'my','him','too', 'y', "you'll", 'were', 'any', 'when', 'and', 'an', 'having', 'after', 'himself', 'is', 'before', 'own','the', 'this', 'down', 'which', 'until', "you're", "she's", 'up', 'ma', 'had', 'how', 'who', 'more', 'with', 'some', 'only', 'it', 'to', 'into', 'both', 'off', 'above', 'll', 'itself', 'against', 'as', 'was', 'while', 'during', 'about'}

colloquial\_dict = {

"bt": "but",

"nver": "never",

"thy": "they",

"bck": "back",

"FB": "Facebook",

"can't": "cannot",

"cnt": "cannot",

"dnt": "do not",

"u": "you",

"ur": "your",

"gr8": "great",

"gonna": "going to",

"gon na": "going to",

"$hit": "shit",

"asap": "as soon as possible",

"e mo masepeng": "is in shit",

"y": "why",

"fokkol": "no",

"pls": "please",

"plz": "please",

"plse": "please",

"fok maan": "damn man",

"u've": "you have",

"uve": "you have",

"hou op lieg asseblief": "please stop lying",

"ppl": "people",

"y'all": "you all",

"Yebo": "yes",

"Voetsek": "go away",

"Ubuntu": "humanity",

"Skebenga": "crook",

"Skelm": "crook",

"Oom": "Older man of authority",

"Oke": "man",

"Now now": "soon",

"Is it": "Is that so",

"Haibo": "wow",

"Gatvol": "fed up",

"Domkop": "idiot",

"Chow": "eat",

"CT": "Cape Town"

}

doc = nlp(text)

for token in doc:

if token.ent\_type\_ == "LOC":

clean\_tokens.append(token.text)

else:

processed\_token = token.text.split("/")

processed\_token = token.text.lower()

processed\_token = re.sub(r'@(\w+)', '', processed\_token)

processed\_token = re.sub(r'http\S+|www\S+|https\S+', '', processed\_token) # remove url

processed\_token = re.sub(r'(?<=[^\w\s])', '', processed\_token) # Remove leading punctuation

processed\_token = re.sub(r'(?=[^\w\s])', '', processed\_token)

processed\_token = emoji.demojize(processed\_token) # remove emojis

processed\_token = processed\_token.replace(":", "") # Remove remaining emoji colons

processed\_token = re.sub(r'\d+|[' + re.escape(string.punctuation) + ']', '', processed\_token) # remove num and punc

if processed\_token in colloquial\_dict:

processed\_token = colloquial\_dict[processed\_token]

corr\_token = spell\_checker.correction(processed\_token)

if processed\_token == "Farramere":

corr\_token = "farramere"

else:

corr\_token = spell\_checker.correction(processed\_token)

if processed\_token == "Limpopo":

corr\_token = "limpopo"

else:

corr\_token = spell\_checker.correction(processed\_token)

if corr\_token == "vial":

print("Correcting 'vial' to 'vaal'")

corr\_token = "vaal"

if corr\_token == "infield":

print("Correcting 'infield' to 'rynfield'")

corr\_token = "rynfield"

if corr\_token == "amp":

print("Correcting 'amp' to ''")

corr\_token = ""

if corr\_token == "jour":

print("Correcting 'jour' to 'johannesburg'")

corr\_token = "johannesburg"

if corr\_token == "gluten":

print("Correcting 'gluten' to 'gauteng'")

corr\_token = "gauteng"

if corr\_token == "jaw":

print("Correcting 'jaw' to 'johannesburg water'")

corr\_token = "johannesburg water"

if corr\_token == "molehill":

print("Correcting 'molehill' to 'morehill'")

corr\_token = "morehill"

if corr\_token == "ear":

print("Correcting 'ear' to 'etr'")

corr\_token = "estimated time of restoration"

if corr\_token == "calamari":

print("Correcting 'calamari to 'kalahari")

corr\_token = "kalahari"

if corr\_token == "northman":

print("Correcting 'northman' to 'northmead'")

corr\_token = "northmead"

if corr\_token == "skim":

print("Correcting 'skim' to 'eskom'")

corr\_token = "eskom"

if corr\_token is not None:

if corr\_token not in stop\_words:

clean\_tokens.append(corr\_token)

lemmatizer = WordNetLemmatizer()

lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in clean\_tokens]

return ' '.join(lemmatized\_tokens)

print(lemmatized\_tokens)

df["clean\_tokens2"] = df["translated\_text"].apply(preprocessor)

df["clean\_tokens2"].unique()