```
#mounting google drive files
from google.colab import drive
import nltk
drive.mount('/content/drive')
nltk.download('punkt')
```

```
Mounted at /content/drive
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
#NLP Text Data folder contains all the folders for the corpuses. The output shows this
import os

# Path to the directory
path = '/content/drive/My Drive/NLP Text Data/'

# List files
files = os.listdir(path)
print(files)
```

```
['machinelearning', 'hiking', 'fishing', 'mathematics']
```

```
#cleaning and prep along with creating the necessary columns
import os
import re
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

nltk.download('stopwords')
stop_words = set(stopwords.words("english"))
stemmer = PorterStemmer()

the_path = "/content/drive/My Drive/NLP Text Data/"

# Clean the original text
def clean_txt(var_in):
    tmp_t = re.sub("[^A-Za-z0-9!@#\$%\^&\*\(\)\-_\+=\.,;:!?'\"]+", " ", var_in).strip()
    return tmp_t

# Remove stopwords from the text
def remove_stopwords(text):
    words = text.split()
    return ' '.join([word for word in words if word.lower() not in stop_words])

# Apply stemming to the text
def apply_stemming(text):
    words = text.split()
    return ' '.join([stemmer.stem(word) for word in words])

# Read and process each file
def read_file(full_path_in):
    with open(full_path_in, "r", encoding="UTF-8") as f_t:
        text_t = f_t.read()  # Reads the entire file
        text_t = clean_txt(text_t)  # Basic cleaning
    return text_t

# Main function to create DataFrame with all columns
def file_crawler(path_in):
    my_pd_t = pd.DataFrame()

    for root, dirs, files in os.walk(path_in, topdown=False):
        for name in files:
            try:
                # Read and clean text
                txt_t = read_file(root + "/" + name)
                if len(txt_t) > 0:
                    the_lab = root.split("/")[-1]

                    # Create different versions of the text
                    body = txt_t
                    body_sw = remove_stopwords(txt_t)
                    body_sw_stem = apply_stemming(body_sw)
```

```
                    # Add to DataFrame
                    tmp_pd = pd.DataFrame({
                        "body": body,
                        "body_sw": body_sw,
                        "body_sw_stem": body_sw_stem,
                        "topic": the_lab
                    }, index=[0])

                    my_pd_t = pd.concat([my_pd_t, tmp_pd], ignore_index=True)

            except Exception as e:
                print(f"Error with file {root}/{name}: {e}")
                pass

    return my_pd_t

# Create the DataFrame
the_data = file_crawler(the_path)
print("Sample of the DataFrame:")
print(the_data.head())  # Print the first few rows of the DataFrame
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Error with file /content/drive/My Drive/NLP Text Data/fishing/UK MongoBD report Nov122018.xls: 'utf-8' codec can't decode byte 0xd0 in p
Error with file /content/drive/My Drive/NLP Text Data/fishing/UK segment count Nov122018.xlsx: 'utf-8' codec can't decode bytes in posit
Error with file /content/drive/My Drive/NLP Text Data/fishing/UK vendor count Nov122108 .xlsx: 'utf-8' codec can't decode bytes in posit
Sample of the DataFrame:
                                                body  \
0  Machine Learning Total 239.99 Computer Science...
1  Rendezvous Server to the Rescue: Dealing with ...
2  The 10 Algorithms Machine Learning Engineers N...
3  Find a Job in Artificial Intelligence or Machi...
4  xkcd: Machine Learning Archive What If? Blag S...

                                             body_sw  \
0  Machine Learning Total 239.99 Computer Science...
1  Rendezvous Server Rescue: Dealing Machine Lear...
2  10 Algorithms Machine Learning Engineers Need ...
3  Find Job Artificial Intelligence Machine Learn...
4  xkcd: Machine Learning Archive If? Blag Store ...

                                         body_sw_stem            topic
0  machin learn total 239.99 comput scienc artifi...  machinelearning
1  rendezv server rescue: deal machin learn logis...  machinelearning
2  10 algorithm machin learn engin need know kdnu...  machinelearning
3  find job artifici intellig machin learn busi i...  machinelearning
4  xkcd: machin learn archiv if? blag store prev ...  machinelearning
```

```
#testing for the token fishing
import re
import pandas as pd

def word_prob(column_name, the_data, token="fishing", decimals=4):
    # Initialize dictionary to store probabilities for the "fishing" token
    probabilities = {"all": None, "fishing": None, "hiking": None, "machinelearning": None, "mathematics": None}

    # Prepare token pattern for matching
    token_pattern = re.escape(token) if " " in token else r'\b' + re.escape(token) + r'\b'

    # Calculate probability for "all" (entire dataset)
    all_tokens = the_data[column_name].str.findall(r'\b\w+\b').apply(len).sum()
    count_token_all = the_data[column_name].str.count(token_pattern).sum()
    probabilities["all"] = round(count_token_all / all_tokens, decimals) if all_tokens > 0 else None

    # Calculate probability for each topic
    topics = ["fishing", "hiking", "machinelearning", "mathematics"]
    for topic in topics:
        topic_df = the_data[the_data["topic"] == topic]
        total_tokens_topic = topic_df[column_name].str.findall(r'\b\w+\b').apply(len).sum()
        count_token_topic = topic_df[column_name].str.count(r'\b' + re.escape(token) + r'\b').sum()
        probabilities[topic] = round(count_token_topic / total_tokens_topic, decimals) if total_tokens_topic > 0 else None

    # Print and return the probabilities dictionary
    print(probabilities)
    return probabilities
```

```
result = word_prob("body", the_data)
```

```
{'all': 0.0012, 'fishing': 0.0057, 'hiking': 0.0001, 'machinelearning': 0.0, 'mathematics': 0.0}
```

```
#testing for the token machine learning
import re
import pandas as pd

def word_prob(column_name, the_data, token="machine learning", decimals=4):
    # Initialize dictionary to store probabilities for the "fishing" token
    probabilities = {"all": None, "fishing": None, "hiking": None, "machinelearning": None, "mathematics": None}

    # Prepare token pattern for matching
    token_pattern = re.escape(token) if " " in token else r'\b' + re.escape(token) + r'\b'

    # Calculate probability for "all" (entire dataset)
    all_tokens = the_data[column_name].str.findall(r'\b\w+\b').apply(len).sum()
    count_token_all = the_data[column_name].str.count(token_pattern).sum()
    probabilities["all"] = round(count_token_all / all_tokens, decimals) if all_tokens > 0 else None

    # Calculate probability for each topic
    topics = ["fishing", "hiking", "machinelearning", "mathematics"]
    for topic in topics:
        topic_df = the_data[the_data["topic"] == topic]
        total_tokens_topic = topic_df[column_name].str.findall(r'\b\w+\b').apply(len).sum()
        count_token_topic = topic_df[column_name].str.count(r'\b' + re.escape(token) + r'\b').sum()
        probabilities[topic] = round(count_token_topic / total_tokens_topic, decimals) if total_tokens_topic > 0 else None

    # Print and return the probabilities dictionary
    print(probabilities)
    return probabilities

result = word_prob("body", the_data)
```

```
{'all': 0.0017, 'fishing': 0.0, 'hiking': 0.0, 'machinelearning': 0.006, 'mathematics': 0.0}
```

```
#testing for the token mathematics
import re
import pandas as pd

def word_prob(column_name, the_data, token="mathematics", decimals=4):
    # Initialize dictionary to store probabilities for the "fishing" token
    probabilities = {"all": None, "fishing": None, "hiking": None, "machinelearning": None, "mathematics": None}

    # Prepare token pattern for matching
    token_pattern = re.escape(token) if " " in token else r'\b' + re.escape(token) + r'\b'

    # Calculate probability for "all" (entire dataset)
    all_tokens = the_data[column_name].str.findall(r'\b\w+\b').apply(len).sum()
    count_token_all = the_data[column_name].str.count(token_pattern).sum()
    probabilities["all"] = round(count_token_all / all_tokens, decimals) if all_tokens > 0 else None

    # Calculate probability for each topic
    topics = ["fishing", "hiking", "machinelearning", "mathematics"]
    for topic in topics:
        topic_df = the_data[the_data["topic"] == topic]
        total_tokens_topic = topic_df[column_name].str.findall(r'\b\w+\b').apply(len).sum()
        count_token_topic = topic_df[column_name].str.count(r'\b' + re.escape(token) + r'\b').sum()
        probabilities[topic] = round(count_token_topic / total_tokens_topic, decimals) if total_tokens_topic > 0 else None

    # Print and return the probabilities dictionary
    print(probabilities)
    return probabilities

result = word_prob("body", the_data)
```

```
{'all': 0.0008, 'fishing': 0.0, 'hiking': 0.0, 'machinelearning': 0.0001, 'mathematics': 0.0036}
```

```
#testing for the token hiking
import re
import pandas as pd
```

```
def word_prob(column_name, the_data, token="hiking", decimals=4):
    # Initialize dictionary to store probabilities for the "fishing" token
    probabilities = {"all": None, "fishing": None, "hiking": None, "machinelearning": None, "mathematics": None}

    # Prepare token pattern for matching
    token_pattern = re.escape(token) if " " in token else r'\b' + re.escape(token) + r'\b'

    # Calculate probability for "all" (entire dataset)
    all_tokens = the_data[column_name].str.findall(r'\b\w+\b').apply(len).sum()
    count_token_all = the_data[column_name].str.count(token_pattern).sum()
    probabilities["all"] = round(count_token_all / all_tokens, decimals) if all_tokens > 0 else None

    # Calculate probability for each topic
    topics = ["fishing", "hiking", "machinelearning", "mathematics"]
    for topic in topics:
        topic_df = the_data[the_data["topic"] == topic]
        total_tokens_topic = topic_df[column_name].str.findall(r'\b\w+\b').apply(len).sum()
        count_token_topic = topic_df[column_name].str.count(r'\b' + re.escape(token) + r'\b').sum()
        probabilities[topic] = round(count_token_topic / total_tokens_topic, decimals) if total_tokens_topic > 0 else None

    # Print and return the probabilities dictionary
    print(probabilities)
    return probabilities

result = word_prob("body", the_data)
```

```
{'all': 0.0007, 'fishing': 0.0, 'hiking': 0.0025, 'machinelearning': 0.0, 'mathematics': 0.0}
```