```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import string
from nltk.stem import PorterStemmer


# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')



# documents defining
document1 = "Mango is a sweet and delicious fruit rich in fiber with many benefits. Mangoes are said to possess many antioxidant properties
document2 = "Flexibility is improved by yoga. Yoga has several benefits. Regular practice of yoga improves muscle strength and posture. Yog
document3 = "Eating bananas will benefit you in different ways. Eat them raw or mixed in your favourite smoothie. To keep the body fit and

# Initialize lemmatizer and stop words
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# Normalise the text to create tokens using suitable preprocessing. Remove stop words and other undesired content.
# Perform Stemming and Lemmatisation.

def preprocess(document):
   # Convert to lowercase
    document = document.lower()

    # Remove punctuation
    document = document.translate(str.maketrans('', '', string.punctuation))

    # Tokenize
    words = word_tokenize(document)

    # Remove stop words and lametize
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]


    return ' '.join(words)

# Preprocess documents
tokens1 = preprocess(document1)
tokens2 = preprocess(document2)
tokens3 = preprocess(document3)
```

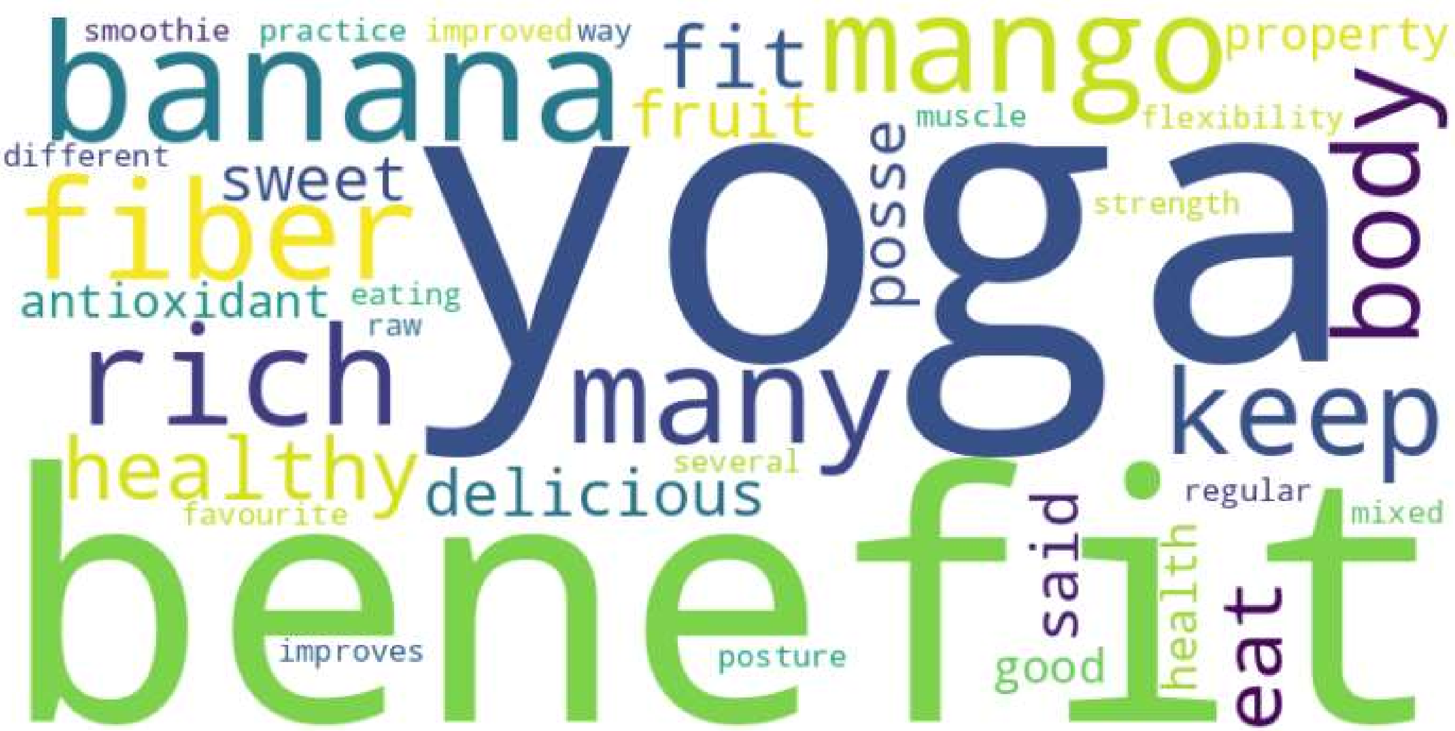
```
import string
from collections import Counter
import matplotlib.pyplot as plt
from wordcloud import WordCloud
# Create a BoW of normalised text. Generate word cloud of BoW.

# Preprocess documents
processed_docs = [preprocess(document1), preprocess(document2), preprocess(document3)]

# Combine all tokens into one string
combined_text = ' '.join(processed_docs)

# Generate and display word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(combined_text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
len(tokens1.split())
tokens2
```

```
'flexibility improved yoga yoga several benefit regular practice yoga improves muscle strength posture yoga keep body healthy fit'
```

```
# Calculate the Term Frequency and TFIDF of the documents. Show the calculation of
# TFIDF of the term “health”.

from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd


# Preprocess documents
processed_docs = [tokens1, tokens2, tokens3]

# Calculate TF and TF-IDF
vectorizer = TfidfVectorizer(use_idf=True, norm='l2')

# Fit and transform the processed documents
tfidf_matrix = vectorizer.fit_transform(processed_docs)

# Convert the TF-IDF matrix to a DataFrame for better readability
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), index=['Document 1', 'Document 2', 'Document 3'], columns=vectorizer.get_feature_names_out(

# Display the TF-IDF DataFrame
print("TF-IDF Matrix:")
print(tfidf_df['health'])

# If you also want to calculate the term frequency (TF) matrix, use the same vectorizer without IDF
tf_vectorizer = TfidfVectorizer(use_idf=False, norm='l2')
tf_matrix = tf_vectorizer.fit_transform(processed_docs)

# Convert the TF matrix to a DataFrame for better readability
tf_df = pd.DataFrame(tf_matrix.toarray(), index=['Document 1', 'Document 2', 'Document 3'], columns=tf_vectorizer.get_feature_names_out())

# Display the TF DataFrame
print("\nTerm Frequency (TF) Matrix:")
print(tf_df['health'])
```

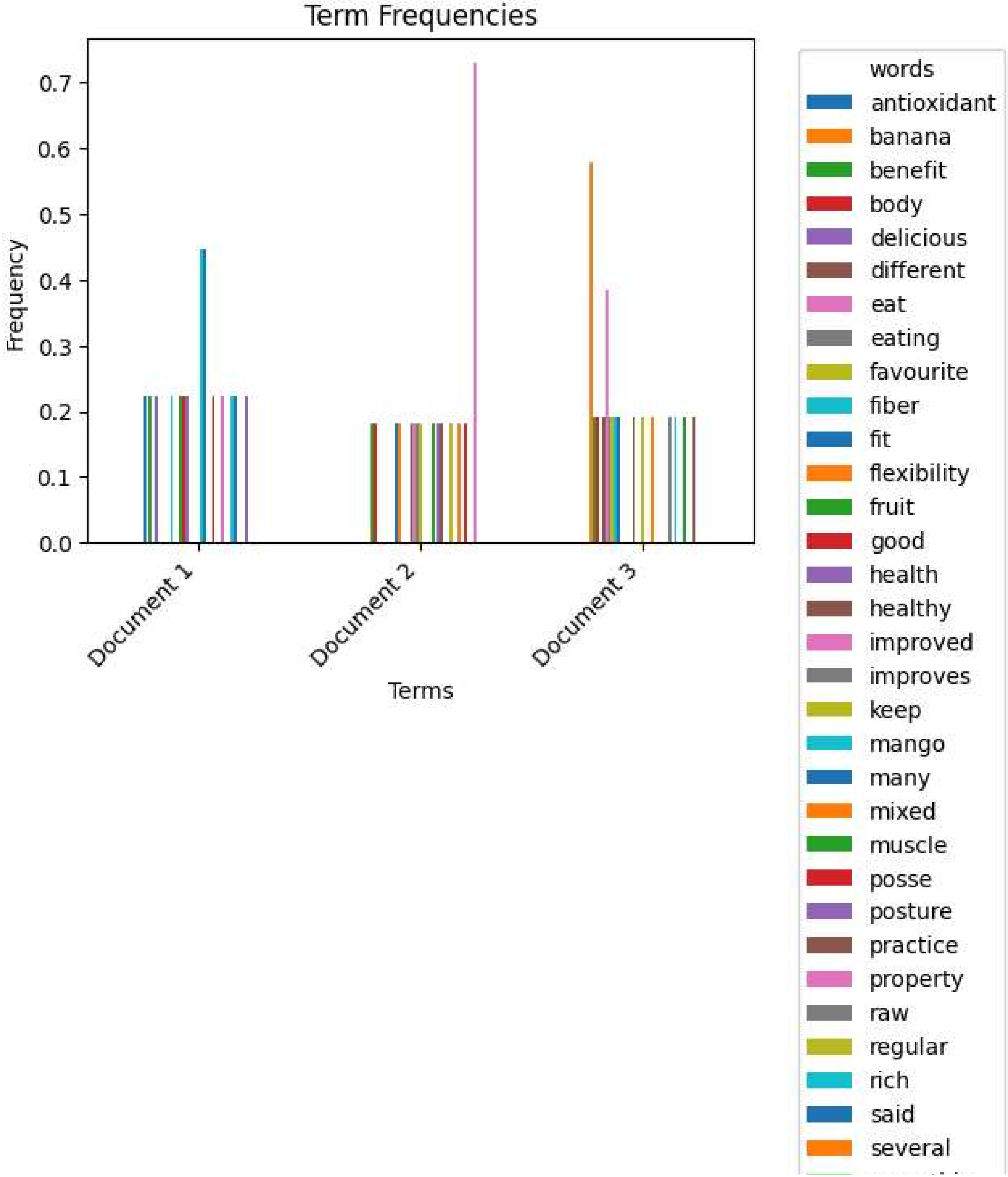
```
TF-IDF Matrix:
Document 1    0.23246
Document 2    0.00000
Document 3    0.00000
Name: health, dtype: float64
```

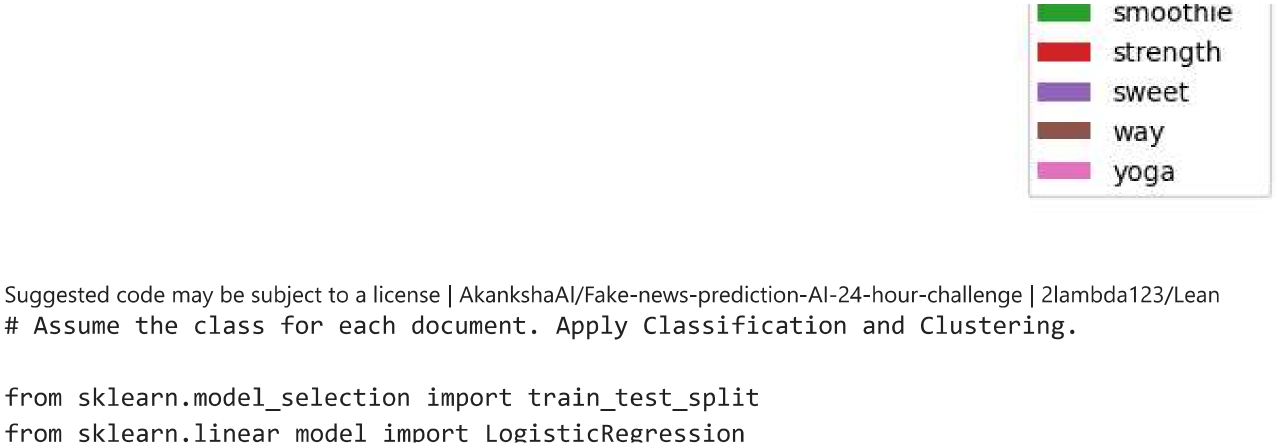
```
Term Frequency (TF) Matrix:
Document 1    0.223607
Document 2    0.000000
Document 3    0.000000
Name: health, dtype: float64
```

```
# Plot the Term Frequencies.
# Plot the Term Frequencies
plt.figure(figsize=(14, 20))
tf_df.head(20).plot(kind='bar', legend=False)
plt.title('Term Frequencies')
plt.xlabel('Terms')
plt.ylabel('Frequency')
plt.tight_layout()  # Adjust layout to fit labels
plt.legend(title='words',loc='upper left', bbox_to_anchor=(1.05, 1))
plt.xticks(rotation=45, ha='right')
# Adjust the position of x-axis labels
# Adjust x-axis labels

# Use `bbox_to_anchor` to move x-axis labels outside the graph area
plt.subplots_adjust(right=0.75)  # Increase right margin
plt.show()
```

```
<Figure size 1400x2000 with 0 Axes>
```


Term Frequencies
Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Document 1
Document 2
Document 3
Terms
words
antioxidant
banana
benefit
body
delicious
different
eat
eating
favourite
fiber
fit
flexibility
fruit
good
health
healthy
improved
improves
keep
mango
many
mixed
muscle
posse
posture
practice
property
raw
regular
rich
said
several



```
# Assume the class for each document. Apply Classification and Clustering.

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```