1. Project Summary:

The objective of this project is to develop a text classification model that can accurately differentiate between scam and legitimate messages based on their content.

About the Dataset: SMS scam Collection

Context

The SMS scam Collection dataset comprises SMS messages categorized as either legitimate (ham) or scam, gathered for research on SMS scam. It contains a dataset consisting of 5,574 SMS messages in English.

Content

Each file in the dataset contains individual messages, with each line comprising two columns:

- `v1` : Represents the label (ham or scam).
- `v2` : Contains the raw text of the message.

Source

The dataset is accessible on Kaggle via the following link: SMS Spam Collection Dataset

2. Data Preparation

Libraries Imported:

1. `numpy` and `pandas` :

- **Purpose:** These libraries are fundamental for data manipulation and analysis.
- **Usage:** They are utilized for handling datasets, such as loading and processing data.

2. `matplotlib.pyplot` and `seaborn` :

- **Purpose:** These libraries are used for data visualization.
- **Usage:** They facilitate the creation of plots and graphs to visualize data distributions and patterns.

3. `nltk` (Natural Language Toolkit):

- **Purpose:** NLTK is a platform for building Python programs to work with human language data.
- **Usage:** It is employed here for text preprocessing tasks, such as tokenization, stemming, and removal of stopwords.

4. `sklearn` (Scikit-learn):

- **Purpose:** Scikit-learn provides simple and efficient tools for data mining and data analysis.
- **Usage:** It is utilized for various machine learning tasks, including feature extraction, model training, and evaluation.

5. `tensorflow.keras` :

- **Purpose:** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.
- **Usage:** It is used here for implementing neural network models for text classification.

6. `wordcloud` :

- **Purpose:** Wordcloud is a visualization technique used for displaying frequently occurring words in text data.
- **Usage:** It helps in gaining insights into the most common words present in the dataset.

7. `termcolor` :

- **Purpose:** Termcolor is used for producing colored output in the terminal.
- **Usage:** It enhances readability by highlighting important messages or notifications.

[1]:

```python
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  # Added Seaborn for visualization

# For Text Preprocessing with NLTK
import re
import string
import nltk  # Added NLTK
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer

# For Machine Learning Models and Evaluation

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer  # Import CountVectorizer from sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# For Neural Network Implementation
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional, GlobalMaxPooling1D, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

# Download NLTK resources if needed
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

from wordcloud import WordCloud

from termcolor import colored  # For colored output

# Informative message confirming successful import of necessary libraries
print(colored("\nNECESSARY LIBRARIES WERE SUCCESSFULLY IMPORTED...", color="blue", attrs=["bold", "dark"]))
```

```
NECESSARY LIBRARIES WERE SUCCESSFULLY IMPORTED...
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kpkum\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\kpkum\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\kpkum\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

2.1 Reading and displaying the first few rows of the CSV File:

- **Purpose:** Read the CSV file containing scam data.
- **Usage:** Utilized Pandas' `read_csv()` function to load the dataset into a DataFrame.
- **Purpose:** Provide a preview of the dataset.
- **Usage:** Used the `head()` method to display the first few rows of the DataFrame.

[2]:

```python
# Read the CSV file containing scam data
data = pd.read_csv("C:/Users/kpkum/Desktop/Kumaravel_FYP/spam.csv", encoding="ISO-8859-1", engine="python")

# Display the first few rows
data.head()
```

[2]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

2.2 Renaming Dataset Columns:

- **Purpose:** Improve column naming for better clarity.
- **Usage:** Renamed columns 'v1' to 'target' and 'v2' to 'text' using the `rename()` method.

[3]:

```python
#rename dataset columns
data.rename(columns = {"v1": "target", "v2": "text"}, inplace = True)

#drop unnecessary columns
data.drop(["Unnamed: 2","Unnamed: 3", "Unnamed: 4"], axis = 1, inplace = True)

# Display the first few rows
data.head(5)
```

[3]:

	target	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

[4]:

```python
# Display concise summary information about the DataFrame 'data'
# including memory usage details and verbosity enabled
data.info(memory_usage=True, verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   target  5572 non-null   object
 1   text    5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

[5]:

```python
# Calculate the number of duplicated values in the dataset and print the count in a colored format
duplicated_count = data.duplicated().sum()  # Count duplicated values in the DataFrame 'data'

# Print the count of duplicated values using colored output
print(colored("There are {} duplicated values in the dataset".format(duplicated_count), color="blue", attrs=["bold", "dark"]))
```

```
There are 403 duplicated values in the dataset
```

2.3 Dropping Unnecessary Columns:

- **Purpose:** Remove irrelevant columns to streamline the dataset.
- **Usage:** Dropped columns 'Unnamed: 2', 'Unnamed: 3', and 'Unnamed: 4' using the `drop()` method.

[6]:

```python
# Remove duplicate rows from the DataFrame 'data' and modify it in place
data.drop_duplicates(inplace=True)
```

[7]:

```python
# Group the data by the 'target' column and count occurrences in each group
grouped_target = data.groupby("target").count()

# Extract the 'text' column from the grouped data
target_text_count = grouped_target["text"]

# Check the count of null values in each column of the DataFrame 'data'
data.isnull().sum()
```

[7]:

```
target    0
text      0
dtype: int64
```

3.Explorartory Data Analysis

In this segment of my project, I had decided to take a deeper look into the scam dataseheet and generate some insigts.

3.1 Identify (ham or scam) frequencies

I wanted to visually represent the distribution of classes ('ham' and 'scam') in the dataset. By creating a bar plot, I aim to provide an overview of the frequency of each class, which helps in understanding the balance of classes in the dataset.

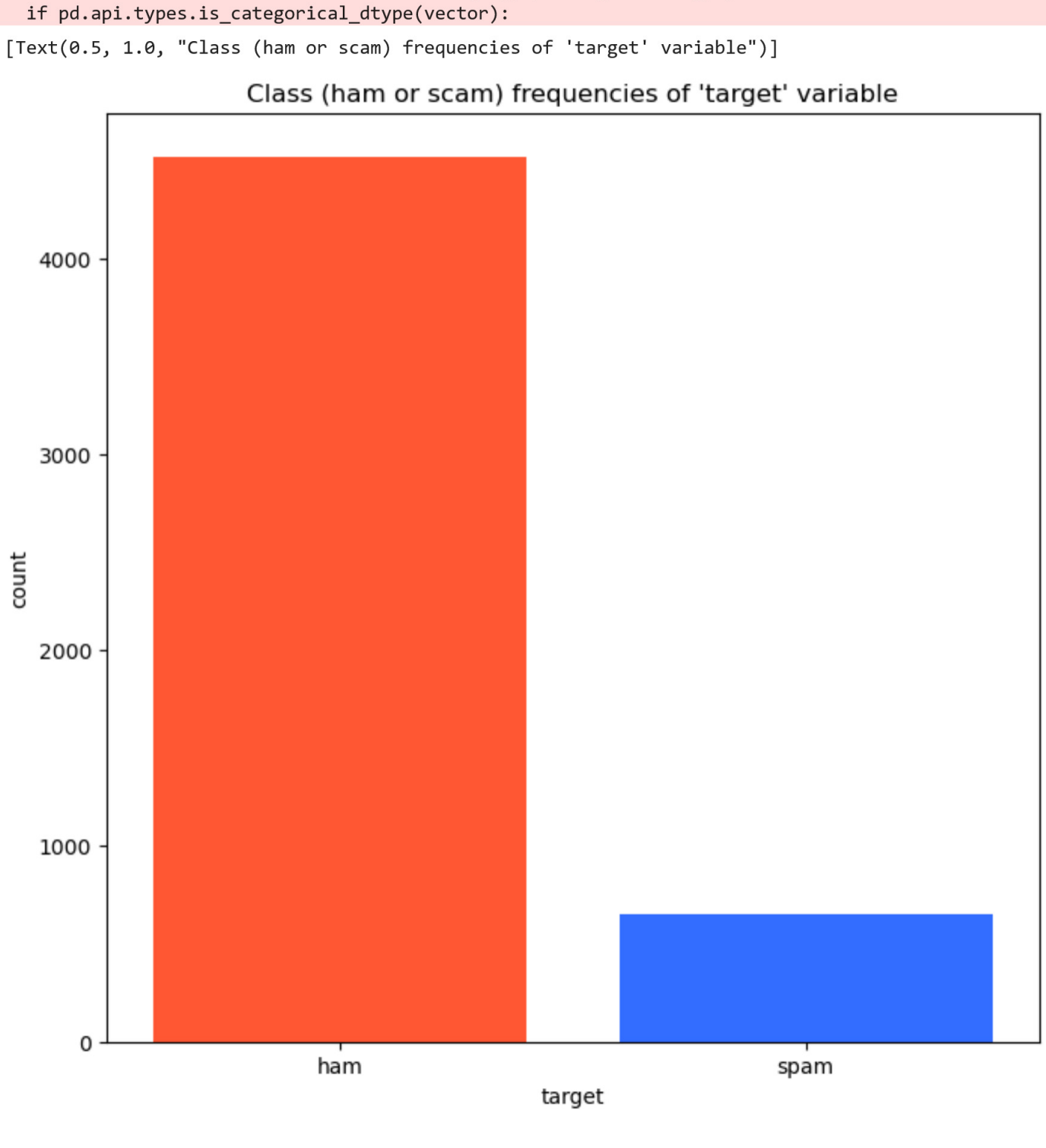
[8]:

```python
# Set up the figure for plotting a bar chart
plt.figure(figsize=[8, 8], clear=True, facecolor="white")

# Define two different colors for the bar plot
colors = ["#FF5733", "#336DFF"]

# Create a bar plot displaying the frequencies of classes in the 'target' variable
sns.barplot(
    x=data["target"].value_counts().index,  # X-axis: class labels
    y=data["target"].value_counts(),  # Y-axis: count of each class
    palette=colors,  # Assign the defined colors to the bars
    saturation=1  # Adjusts the intensity of colors
).set(
    title="Class (ham or scam) frequencies of 'target' variable"  # Set the plot title
)
```

```
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

[8]:

```
[Text(0.5, 1.0, "Class (ham or scam) frequencies of 'target' variable")]
```



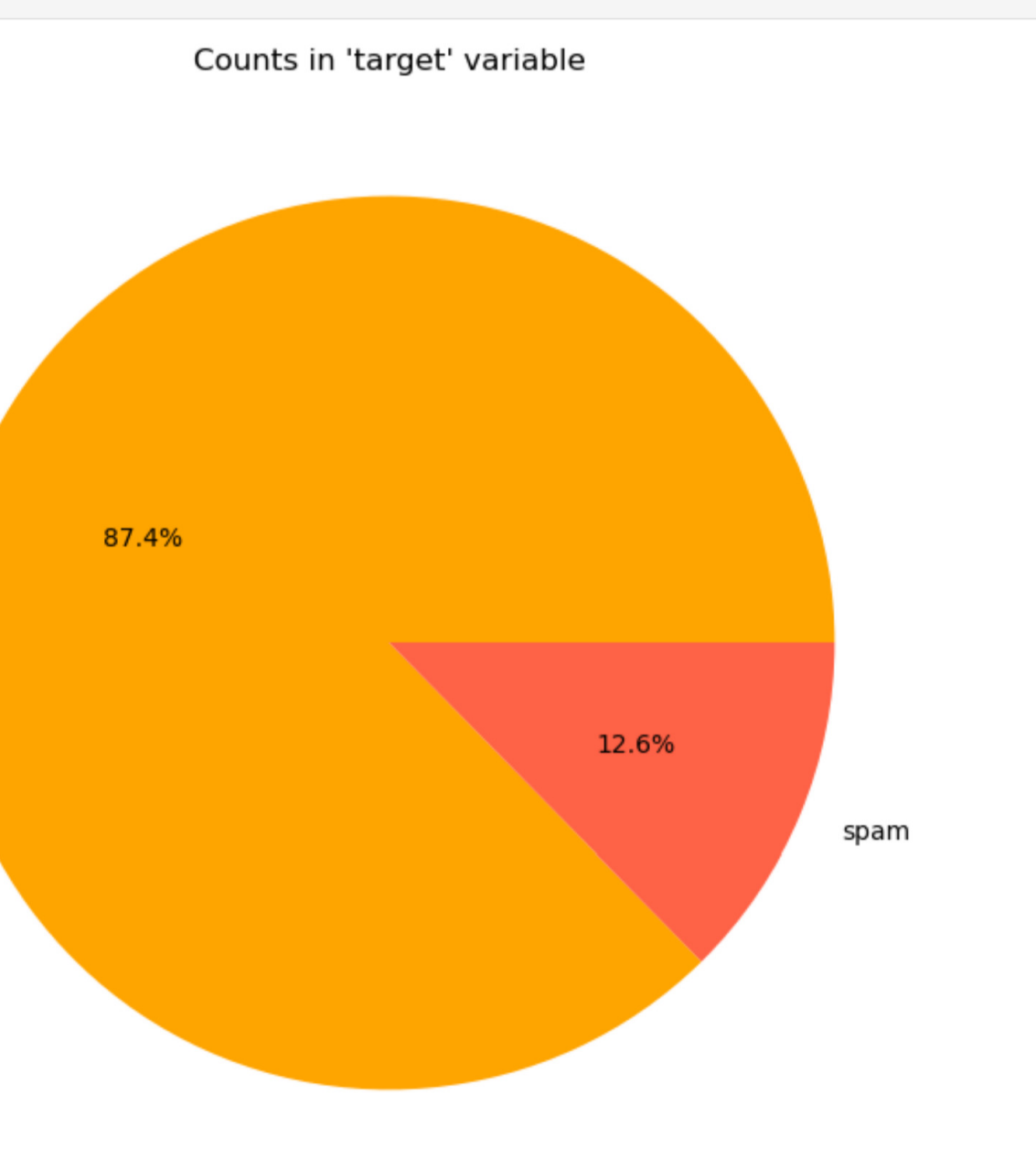
Class (ham or scam) frequencies of 'target' variable

3.2 Identify Counts of ham and scam messages

I wanted to visualize the distribution of classes ('ham' and 'scam') in the 'target' variable using a pie chart. This chart allows for a clear representation of the proportion of each class within the dataset.

[9]:

```python
# Calculate value counts for 'target' column
target_counts = data['target'].value_counts()

# Create a pie chart using Matplotlib
plt.figure(figsize=(8, 8))
plt.pie(target_counts, labels=target_counts.index, autopct='%1.1f%%', colors=['#FFA500', '#FF6347'])
plt.title("Counts in 'target' variable")
plt.show()
```



Counts in 'target' variable

3.3 Identify character count, word count and sentence count and generate stats

I wanted to transform the text data in the 'text' column by calculating the character count, word count, and sentence count for each text. Additionally, descriptive statistics are generated for the added columns to provide insights into the distribution of these features within the dataset.

[10]:

```python
# Calculate the character count for each text in the 'text' column and store it in a new column 'ch_no'
data["ch_no"] = data["text"].apply(len)

# Calculate the word count for each text using NLTK word_tokenize and store it in a new column 'wd_no'
data["wd_no"] = data.apply(lambda row: nltk.word_tokenize(row["text"]), axis=1).apply(len)

# Calculate the sentence count for each text using NLTK sent_tokenize and store it in a new column 'sn_no'
data["sn_no"] = data.apply(lambda row: nltk.sent_tokenize(row["text"]), axis=1).apply(len)

# Generate descriptive statistics for the added columns and apply background color in a orange gradient style
data.describe().T.style.background_gradient(cmap="Oranges")
```

[10]:

	count	mean	std	min	25%	50%	75%	max
ch_no	5169.000000	78.977945	58.236293	2.000000	36.000000	60.000000	117.000000	910.000000
wd_no	5169.000000	18.455794	13.324758	1.000000	9.000000	15.000000	26.000000	220.000000
sn_no	5169.000000	1.965564	1.448541	1.000000	1.000000	1.000000	2.000000	38.000000

3.4 Generating pairplot

Create a pairplot using Seaborn to visualize pairwise relationships in the dataset

[11]:

```python
# Create a pairplot using Seaborn to visualize pairwise relationships in the dataset
fig = sns.pairplot(
    data=data,  # Data to be plotted
    hue="target",  # Variable for coloring the plot
    palette=["#FF5733", "#FFA500"],  # Define the palette colors (red and orange)
    height=3.15,  # Height of each subplot
    aspect=1.35  # Aspect ratio of the plots
)

# Show the pairplot
plt.show(fig)
```

```
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

3: Data Preprocessing

- Cleaned the text data by eliminating punctuation, URLs, and stopwords.
- Applied stemming and tokenization to prepare the messages for analysis.

[12]:

```python
def clean_text(text):
    text = str(text).lower()  # Convert text to lowercase
    text = re.sub('\[.*?\]', '', text)  # Remove text within square brackets
    text = re.sub('https?://\S+|www\.\S+', '', text)  # Remove URLs
    text = re.sub('<.*?>+', '', text)  # Remove HTML tags
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)  # Remove punctuation
    text = re.sub('\n', '', text)  # Remove newline characters
    text = re.sub('\w*\d\w*', '', text)  # Remove alphanumeric words with digits
    return text
```

[13]:

```python
# Assuming 'data' is your DataFrame
data['text_clean'] = data['text'].apply(clean_text)

# Keeping only 'text_clean' and 'target' columns
data = data[['text_clean', 'target']]
data.head()
```

[13]:

	text_clean	target
0	go until jurong point crazy available only in ...	ham
1	ok lar joking wif u oni	ham
2	free entry in a wkly comp to win fa cup final...	spam
3	u dun say so early hor u c already then say	ham
4	nah i dont think he goes to usf he lives aroun...	ham

3.1 preprocess the text data by removing stopwords,

Stopwords are commonly occurring words that do not typically contribute to the meaning of the text. The provided code implements a function to remove stopwords from the 'text_clean' column in the dataset.

[14]:

```python
stop_words = stopwords.words('english')  # Get English stopwords

more_stopwords = ['u', 'im', 'c']  # Additional stopwords

stop_words = stop_words + more_stopwords  # Combine all stopwords

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

data['text_clean'] = data['text_clean'].apply(remove_stopwords)  # Apply removal function to 'text_clean' column
data.head()  # Display updated 'text_clean' data
```

```
C:\Users\kpkum\AppData\Local\Temp\ipykernel_6268\532766798.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text_clean'] = data['text_clean'].apply(remove_stopwords)  # Apply removal function to 'text_clean' column
```

[14]:

	text_clean	target
0	go jurong point crazy available bugis n great ...	ham
1	ok lar joking wif oni	ham
2	free entry wkly comp win fa cup final tkts m...	spam
3	dun say early hor already say	ham
4	nah dont think goes usf lives around though	ham

3.2 further preprocess the text data

By applying stemming, a technique used to reduce words to their root or base form. The provided code utilizes the Snowball stemmer for English to stem words in the 'text_clean' column of the dataset.

[15]:

```python
stemmer = nltk.SnowballStemmer("english")  # Initialize Snowball stemmer for English

def stemm_text(text):
    # Stem words in text
    text = ' '.join(stemmer.stem(word) for word in text.split(' '))
    return text
```

[16]:

```python
data['text_clean'] = data['text_clean'].apply(stemm_text)  # Apply stemming function to 'text_clean' column
data.head()  # Display the first few rows of the updated 'text_clean' data
```

```
C:\Users\kpkum\AppData\Local\Temp\ipykernel_6268\1116890594.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text_clean'] = data['text_clean'].apply(stemm_text)  # Apply stemming function to 'text_clean' column
```

[16]:

	text_clean	target
0	go jurong point crazi avail bugi n great world...	ham
1	ok lar joke wif oni	ham
2	free entri wkli comp win fa cup final tkts m...	spam
3	dun say earli hor alreadi say	ham
4	nah dont think goe usf live around though	ham

3.3 Developing a preprocessing function that can be used on the dataset

Below I have developed a function `preprocess_data(text)` which helps to perform a series of preprocessing steps (based on the steps provided above) on input text data. These steps include cleaning punctuation and URLs, removing stopwords, and stemming the remaining words.

[17]:

```python
def preprocess_data(text):
    # Clean punctuation, URLs, and so on
    text = clean_text(text)
    # Remove stopwords
    text = ' '.join(word for word in text.split(' ') if word not in stop_words)
    # Stem all the words in the sentence
    text = ' '.join(stemmer.stem(word) for word in text.split(' '))

    return text
```

[18]:

```python
data['text_clean'] = data['text_clean'].apply(preprocess_data)  # Apply preprocess_data function to 'text_clean' column
data.head()  # Display the first few rows of the updated 'text_clean' data
```

```
C:\Users\kpkum\AppData\Local\Temp\ipykernel_6268\2447491990.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text_clean'] = data['text_clean'].apply(preprocess_data)  # Apply preprocess_data function to 'text_clean' column
```

[18]:

	text_clean	target
0	go jurong point crazi avail bugi n great world...	ham
1	ok lar joke wif oni	ham
2	free entri wkli comp win fa cup final tkts m...	spam
3	dun say ear hor alreadi say	ham
4	nah dont think goe usf live around though	ham

3.4 Labeling the data

I had then perform label encoding on the categorical target variable ('target') in the dataset. Label encoding is a preprocessing technique used to convert categorical labels into numerical format, which is required by many machine learning algorithms.

[19]:

```python
le = LabelEncoder()  # Initialize LabelEncoder

le.fit(data['target'])  # Fit LabelEncoder to 'target' column

data['target_encoded'] = le.transform(data['target'])  # Transform 'target' column and assign to 'target_encoded'
data.head()  # Display the first few rows including the 'target_encoded' data
```

```
C:\Users\kpkum\AppData\Local\Temp\ipykernel_6268\2507877696.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['target_encoded'] = le.transform(data['target'])  # Transform 'target' column and assign to 'target_encoded'
```

[19]:

	text_clean	target	target_encoded
0	go jurong point crazi avail bugi n great world...	ham	0
1	ok lar joke wif oni	ham	0
2	free entri wkli comp win fa cup final tkts m...	spam	1
3	dun say ear hor alreadi say	ham	0
4	nah dont think goe usf live around though	ham	0

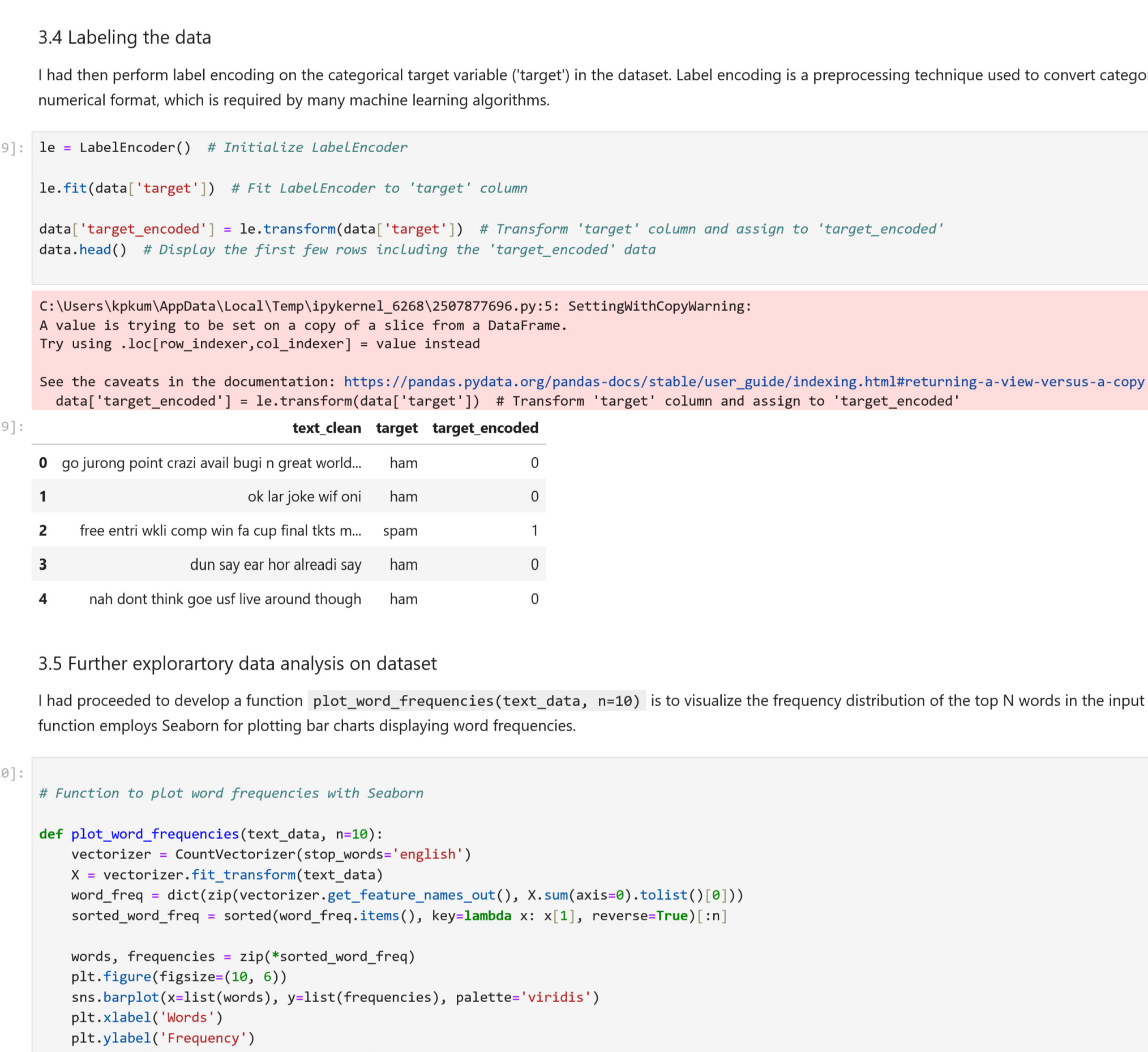
3.5 Further explorartory data analysis on dataset

I had proceeded to develop a function `plot_word_frequencies(text_data, n=10)` is to visualize the frequency distribution of the top N words in the input text data. This function employs Seaborn for plotting bar charts displaying word frequencies.

[20]:

```python

# Function to plot word frequencies with Seaborn

def plot_word_frequencies(text_data, n=10):
    vectorizer = CountVectorizer(stop_words='english')
    X = vectorizer.fit_transform(text_data)
    word_freq = dict(zip(vectorizer.get_feature_names_out(), X.sum(axis=0).tolist()[0]))
    sorted_word_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)[:n]

    words, frequencies = zip(*sorted_word_freq)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=list(words), y=list(frequencies), palette='viridis')
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.title(f'Top {n} Word Frequencies')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


text_data = data['text_clean']

# Plotting top N word frequencies with Seaborn
plot_word_frequencies(text_data, n=20)  # Change 'n' to display more or fewer top words
```

```
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: unique with argument that is not not a Series,
Index, ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
  order = pd.unique(vector)
C:\Users\kpkum\anaconda3\envs\kumaravel_FYP_2023\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



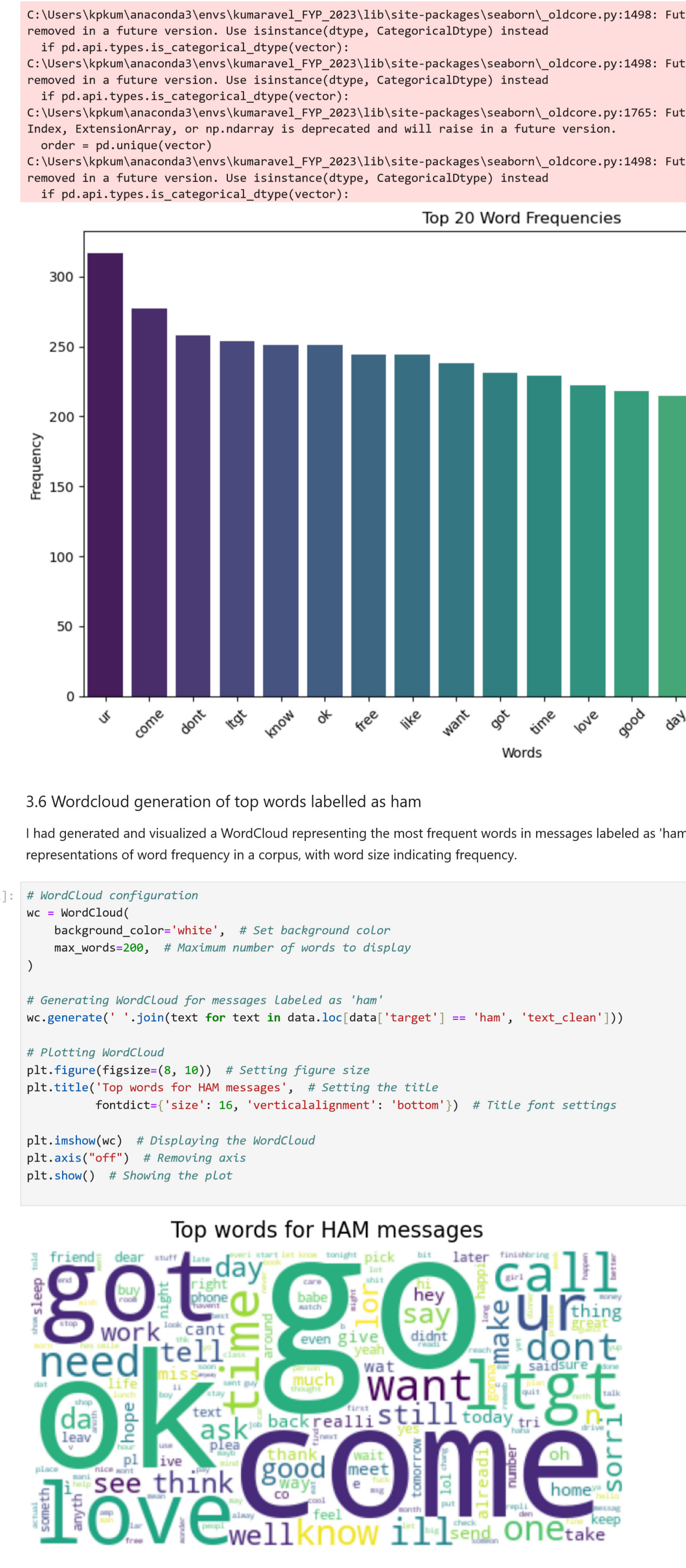
Top 20 Word Frequencies

3.6 Wordcloud generation of top words labelled as ham

I had generated and visualized a WordCloud representing the most frequent words in messages labeled as 'ham' (non-scam) in the dataset. WordClouds are graphical representations of word frequency in a corpus, with word size indicating frequency.

[21]:

```python
# WordCloud configuration
wc = WordCloud(
    background_color='white',  # Set background color
    max_words=200,  # Maximum number of words to display
)

# Generating WordCloud for messages labeled as 'ham'
wc.generate(' '.join(text for text in data.loc[data['target'] == 'ham', 'text_clean']))

# Plotting WordCloud
plt.figure(figsize=(8, 10))  # Setting figure size
plt.title('Top words for HAM messages',  # Setting the title
          fontdict={'size': 16, 'verticalalignment': 'bottom'})  # Title font settings

plt.imshow(wc)  # Displaying the WordCloud
plt.axis("off")  # Removing axis
plt.show()  # Showing the plot
```



Top words for HAM messages