PHASE – 5

NAME	**HEMANATH M D**
REGISTER NUMBER	**61772221L01**
NM_ID	**aut2221l301**
PROJECT TITLE	**BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER**

PROBLEM DEFINITION:

The problem is to build an AI-powered spam classifier that can accurately distinguish between spam and non-spam messages in emails or text messages. The goal is to reduce the number of false positives (classifying legitimate messages as spam) and false negatives (missing actual spam messages) while achieving a high level of accuracy.

DESIGN THINKING:

STEP 1: DATA COLLECTION

- For the purpose of data collection, we can explore the availability of a dataset containing labelled samples of both spam and non-spam messages, which may be sourced from platforms like Kaggle.
- Ensure the dataset is diverse.

STEP 2: DATA PRE-PROCESSING

The text data needs to be cleaned and pre-processed. This involves removing special characters, converting text to lowercase, and tokenizing the text into individual words.

Text Data Cleaning:

This involves the removal of any unwanted or irrelevant characters from the text data. Special characters, symbols, and formatting that do not contribute to the understanding of the text are typically eliminated. For example, removing punctuation marks, HTML tags, or non-alphanumeric characters.

Converting Text to Lowercase:

Text data is often converted to lowercase to ensure uniformity in text analysis. This step helps in treating words in different cases (e.g., "Hello" and "hello") as the same, preventing potential duplication and improving the consistency of text features.

Tokenization:

Tokenization is the process of splitting the text into individual words or tokens. This step is crucial for natural language processing (NLP) tasks as it breaks down the text into manageable units for analysis. For example, the sentence "I love machine learning" would be tokenized into ["I", "love", "machine", "learning"].

STEP 3: FEATURE EXTRACTION

- Convert the tokenized words into numerical features that the neural network can understand.
- Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency), which quantifies the importance of words in the documents.
- This step transforms raw text into a format suitable for deep learning models to process.

STEP 4: MODEL SELECTION

- Choose a deep learning architecture for the spam classifier. Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units are suitable for sequential data like text.
- Design the neural network with embedding layers to represent words, LSTM layers to capture sequential patterns, and output layers with sigmoid activation for binary classification (spam or non-spam).

STEP 5: EVALUATION

We will measure the model's performance using metrics like accuracy, precision, recall, and F1-score.

Accuracy:

This metric tells us the percentage of messages the model correctly classified as either spam or non-spam. It's a measure of overall correctness.

Precision:

Precision tells us the proportion of messages the model classified as spam that were actually spam. In other words, it measures how many of the predicted spam messages were truly spam.

Recall:

Recall, also known as sensitivity, tells us the proportion of actual spam messages that the model correctly identified as spam. It measures the model's ability to find all the spam messages.

F1-Score:

The F1-score is a single number that combines both precision and recall. It provides a balanced measure of a model's performance, especially when you want to avoid either too many false positives (precision) or too many false negatives (recall).

STEP 6: ITERATIVE IMPROVEMENT

- We will fine-tune the model and experiment with hyperparameters to improve its accuracy.
- Fine-tuning the model involves making adjustments to the model's internal settings or architecture to improve its accuracy and effectiveness in classifying spam and non-spam messages.

- Hyperparameter Tuning: Fine-tuning hyperparameters, which are configuration settings that affect the model's behaviour, to find the best combination for optimal performance.
- Additionally, consider techniques like early stopping to prevent overfitting and optimize the model's performance continually.
- Iterate on the model design and hyperparameters until the desired accuracy and balance between false positives and false negatives are achieved.

ABOUT DATASET

Context

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according being ham (legitimate) or spam.

Content

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

This corpus has been collected from free or free for research sources at the Internet:

- A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.
- A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

- A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis available at [WebLink].
- Finally, we have incorporated the SMS Spam Corpus v.0.1 Big. It has 1,002 SMS ham messages and 322 spam messages and it is public available at: [Web Link]. This corpus has been used in the following academic researches.

Dataset Link: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

IMPORT DATASET :

Importing a dataset means bringing external data into our programming environment, typically in a format that allows us to work with and analyse that data using our programming language.

- **Choose a Programming Language:** we typically use a programming language to work with datasets. Python is a popular choice for data analysis.
- **Select a Data Format:** Datasets can come in various formats, such as CSV, Excel, JSON, SQL databases, or more specialized formats like HDF5. we need to know the format of your dataset to choose the right method for importing it. Here we using CSV dataset format.
- **Install Necessary Libraries:** Depending on the format of our dataset, we might need to install specific libraries to handle it.
- **Import Required Libraries:** Import the necessary libraries into our code using import.

1) Importing the Dataset

In [41]:
```
# import the necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
from nltk.corpus import stopwords
import string
from nltk.stem.porter import PorterStemmer
ps=PorterStemmer()
from wordcloud import WordCloud
from collections import Counter
```

In [42]:
```
# import the dataset

df=pd.read_csv("spam.csv")
```

- “import pandas as pd” and “import seaborn as sns” are lines that bring in two helpful tools for working with data.
- “df = pd.read_csv("spam.csv")” reads a file named "spam.csv" (dataset) and stores it in a table-like structure called a DataFrame.
- “print(df)” shows us what's inside the table. It's like looking at the first few rows of your data so we can understand what's in there.
- So, this code is all about getting data from a file, organizing it in a table, and taking a peek at the data to see what's in it.

DATA CLEANING :

The purpose of data cleaning is to make our data more reliable and usable:

- **Remove Errors:** Fix wrong or unrealistic values in the dataset.
- **Fill the Missing Information:** If some data is missing, we can estimate or find the missing values so our data is complete.
- **Make it Consistent:** If we have the same information in different formats, like "New York" and "NY," we can make it all look the same.
- **Remove Duplicates:** Sometimes, the same information appears more than once. Cleaning helps us to remove duplicates so we don't count things twice.

By doing this, data cleaning ensures that the data we work with is accurate, trustworthy, and ready for analysis.

The dataset shall be cleaned through the following processes:

- Checking the number of columns.
- Changing misspelt column names to the correct names.
- Checking for missing values.
- Checking for Duplicate values.

2) Data Cleaning

```
In [43]: # Get the information about dataset

         df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

```
In [44]: # Drop the unnecessary columns in the dataset

         df.drop(columns=["Unnamed: 2","Unnamed: 3","Unnamed: 4"],inplace=True)
```

```
In [7]: df
Out[7]:
```

	v1	v2
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

```
In [45]: # Rename the columns

         df.rename(columns={'v1':'Type','v2':'Message'},inplace=True)
```

```
In [46]: df
Out[46]:
```

	Type	Message
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
In [47]: # Encode the 'Type' column

         df['Type']=encoder.fit_transform(df['Type'])
```

- 'df.columns' is like looking at the labels on a set of drawers. It shows us the names of the different parts (columns) in our dataset. In this case, there are columns named " Type," " Message," and some others that don't have clear names.
- df.rename(columns={"v1":"Type"},inplace=True)&df.rename(columns={"v2":"Message"}, inplace=True) are like relabeling our drawers so that they have more meaningful names. Instead of "v1" and "v2," now we have " Type" and " Message."
- After renaming, df.columns shows us the updated labels of our drawers.
- df.isnull().sum() is like checking if there's anything missing in our drawers. It tells us how many missing (empty) values there are in each drawer. In this case, "Type" and "Message" have no missing values, but the other drawers have lots of missing stuff.

```
In [18]: # Drop the columns 'unnamed: 2' 'unnamed: 3 'unnamed: 4'
         df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],inplace=True)
```

```
In [15]: df.columns
Out[15]: Index(['Mail_Type', 'Mail_Message'], dtype='object')

In [16]: df.isnull().sum()
Out[16]: Mail_Type       0
         Mail_Message    0
         dtype: int64

In [19]: # checking for information concerning the dataset
         df.info()

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

- **df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True):** Think of our dataset as a collection of drawers. This line of code removes (or drops) some drawers labeled 'Unnamed: 2,' 'Unnamed: 3,' and 'Unnamed: 4.' It's like getting rid of drawers we don't need.

- **df.columns:** This line shows us the labels on our remaining drawers. After the drop, we have only two drawers left, 'Type' and 'Message.'

- **df.isnull().sum():** It's like counting how many empty spaces are inside your remaining drawers. In this case, both 'Type' and 'Message' have no missing (empty) values.

- **df.info():** This provides detailed information about our dataset, like its size and what's inside. It tells us that we have a DataFrame with two columns, 'Type' and 'Message,' and no missing values. It also tells us the memory usage (how much space it takes in your computer's memory).

```
In [49]: # check for missing values

         df.isnull().sum()

Out[49]: Type       0
         Message    0
         dtype: int64

In [50]: # Check for duplicate values

         df.duplicated().sum()

Out[50]: 403

In [51]: # Remove duplicate values

         df=df.drop_duplicates(keep='first')

In [52]: # Recheck for duplicates

         df.duplicated().sum()

Out[52]: 0

In [53]: # Check the dimensions of the dataframe

         df.shape

Out[53]: (5169, 2)
```

- **df["Type"].unique():** This line is like looking at a specific drawer labeled "Mail_Type" in our data. It shows us all the different things (unique values) we find in that drawer. In this case, there are two unique values: 'ham' and 'spam.'
- **df.duplicated():** It's like checking if any information in our data is repeated. It tells us for each row in our data if it's a duplicate (a repeat) or not. If it's not a duplicate, it says 'False,' and if it is a duplicate, it says 'True.'
- **df.drop_duplicates():** we're making a new collection of drawers (a new dataset) by removing the duplicates. It's like taking out repeated pieces of information from our data.
- **df.duplicated().sum():** This checks if there are any duplicates left in our new dataset (the new collection of drawers). If the sum is zero, it means there are no duplicates left.
- **df.to_csv("cleaned_spam.csv", index=False):** It's like taking our cleaned data and putting it in a new file named "cleaned_spam.csv." The "index=False" part just means we don't want to include the row numbers when saving the data

DATA PREPROCESSING:

```
In [70]: # Function for preprocess the dataset

def transform_text(text):
    # Convert the message to lower case
    text=text.lower()

    # Tokenize the message
    text=nltk.word_tokenize(text)

    # Remove the special characters in the message
    y=[]
    for i in text:
        if i.isalnum():
            y.append(i)
    text=y[:]
    y.clear()

    # Remove the stop words and punctuations in the message
    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    text=y[:]
    y.clear()

    # Stemming
    for i in text:
        y.append(ps.stem(i))

    return " ".join(y)

df['transformed_text']=df['Message'].apply(transform_text)
```

```
In [72]: df
```

Out[72]:

	Type	Message	num_of_characters	num_of_words	num_of_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah think goe usf live around though
...	...	...	...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	161	35	4	2nd time tri 2 contact u pound prize 2 claim e...
5568	0	Will Ì_ b going to esplanade fr home?	37	9	1	b go esplanad fr home
5569	0	Pity, * was in mood for that. So...any other s...	57	15	2	piti mood suggest
5570	0	The guy did some bitching but I acted like i'd...	125	27	1	guy bitch act like interest buy someth els nex...
5571	0	Rofl. Its true to its name	26	7	2	rofl true name

5169 rows × 6 columns

```
In [73]: # Writing to a csv file

df.to_csv("processed_dataset.csv")
```

1. Cleaning the Text:

- import re brings in a library for text manipulation.
- clean_text(text) is a function that does several things to make text consistent:

  text = text.lower() makes all text lowercase.

- re.sub(r"[^a-zA-Z0-9]", " ", text) removes anything that isn't a letter or number and replaces it with a space.
- df['Message'].apply(clean_text) applies this cleaning to each message in the 'Message' column in our dataset, making the text consistent and easier to work with.
- print(df) displays the cleaned dataset.

2. Natural Language Processing (NLP) Preprocessing:

- !pip install nltk installs the Natural Language Toolkit (NLTK) library.
- import nltk and nltk.download('punkt') load and download necessary data for NLTK.
- from nltk.corpus import stopwords and nltk.download('stopwords') import a list of common words (stopwords) in English and download related data.
- ps = PorterStemmer() creates a stemmer, which reduces words to their base form.

3. Text Transformation:

- transfrom_text(text) is a function that processes text:
- text = nltk.word_tokenize(text) splits the text into words.
- A loop filters out special characters and non-alphanumeric characters.
- Another loop removes stopwords (common words like "the" and "and") and punctuation.
- Yet another loop performs stemming to reduce words to their base form (e.g., "running" becomes "run").
- The final result is a cleaned and transformed text.
- df['Message'].apply(transfrom_text) applies this text transformation to each message in the 'Message' column.
- print(df) shows the updated dataset with the transformed text.

4. Saving the Preprocessed Data:

- df.to_csv('preprocessed_dataset.csv') saves the preprocessed dataset to a new CSV file called 'preprocessed_dataset.csv.' This file contains the cleaned and transformed text for further analysis.

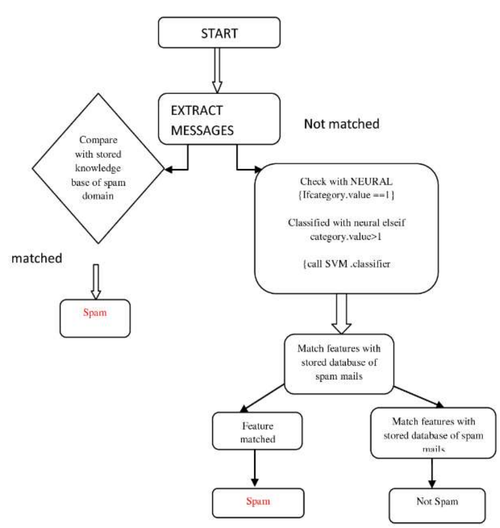
DATA ANALYSIS:

Model : Neural Networks

Data analysis using neural networks is like teaching a computer to recognize and understand patterns in data, just like how we humans recognize things.

- **Data Input:** Imagine we have a lot of emails - some are spam (unwanted junk emails) and some are not. Each email is like a piece of data.
- **Learning:** we use a neural network to teach our computer to recognize patterns in these emails. For instance, it learns that spam emails often contain words like "buy now," "free," or "win."
- **Recognition:** After learning from the data, the computer can look at a new email and decide whether it's spam or not. If it sees lots of words like "buy now" and "free," it might say, "This is likely spam."
- **Improvement:** The more emails we give it, the better it gets at recognizing spam. It learns to spot more patterns and becomes better at avoiding false alarms (marking non-spam as spam) or missing real spam.
- **Applications:** we can use this spam classifier to automatically filter out spam emails from our inbox, keeping our email safe and organized.

The purpose of this data analysis with a neural network is to save our time by automatically identifying and filtering out annoying and potentially harmful spam emails. It's like having a smart assistant who's really good at spotting and getting rid of junk mail for us.



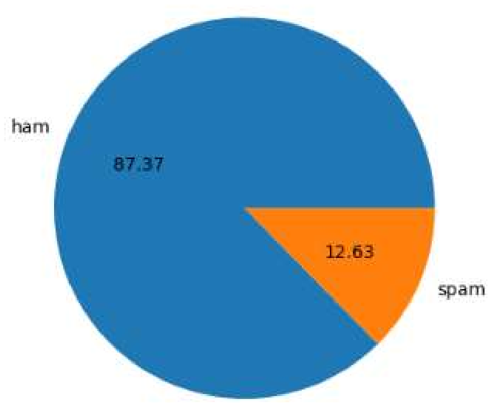
The dataset shall be analysed through the following processes:

- Exploratory Data Analysis (EDA).
- Length of the Message
  - Num_characters
  - Num_words
  - Num_sentence
- Word Frequency Analysis.
- Message length Analysis.

Exploratory Data Analysis (EDA):

Checking ham and spam count:

```
In [54]: # Count the values of 'Type' column
         df['Type'].value_counts()

Out[54]: Type
         0    4516
         1     653
         Name: count, dtype: int64

In [55]: # Plot a pie chart for 'Type' column

         plt.pie(df['Type'].value_counts(),labels=['ham','spam'],autopct="%0.2f")

Out[55]: ([<matplotlib.patches.Wedge at 0x11592292bd0>,
           <matplotlib.patches.Wedge at 0x11591ec45d0>],
          [Text(-1.0144997251399075, 0.4251944351600247, 'ham'),
           Text(1.014499764949479, -0.4251943401757036, 'spam')],
          [Text(-0.5533634864399495, 0.23192423736001344, '87.37'),
           Text(0.5533635081542612, -0.23192418555038377, '12.63')])
```



Code Explanation:

- df['Type'].value_counts(): This line is like counting how many times we find different things in a drawer labeled " Type." In this case, there are 4,516 emails labeled "ham" (non-spam) and 653 emails labeled "spam."
- # EDA: This comment suggests that we're moving into Exploratory Data Analysis (EDA). EDA is like taking a closer look at your data to understand it better.
- !pip install matplotlib: This line installs a library called Matplotlib, which is used for creating visualizations like charts and graphs.
- plt.pie(df['Type'].value_counts(), labels=['ham', 'spam'], autopct="%0.2f"): This code creates a pie chart. It takes the counts we found in step 1 and represents them in a pie chart. The "ham" slice is larger because there are more "ham" emails. The "autopct" part adds percentage labels to the chart.

Finding Length of the Message:

1. **Checking number of characters:**

```
In [56]: # Count the number of characters in a message
         df['num_of_characters']=df['Message'].apply(len)
```

2. **Checking number of Words:**

```
In [58]: # Count the number of words in a message
         df['num_of_words']=df['Message'].apply(lambda x:len(nltk.word_tokenize(x)))
```

3. **Checking number of Sentence:**

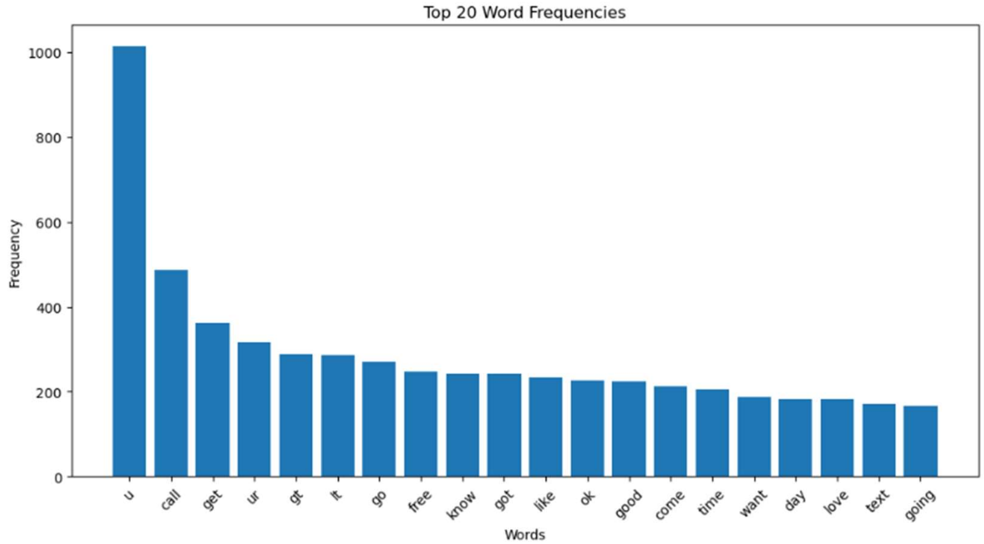
```
In [61]: # Count the number of sentences in a message
         df['num_of_sentences']=df['Message'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

Out[62]:

	Type	Message	num_of_characters	num_of_words	num_of_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1
...	...	...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	161	35	4
5568	0	Will ì_ b going to esplanade fr home?	37	9	1
5569	0	Pity, * was in mood for that. So...any other s...	57	15	2
5570	0	The guy did some bitching but I acted like i'd...	125	27	1
5571	0	Rofl. Its true to its name	26	7	2

5169 rows × 5 columns

Word Frequency Analysis:

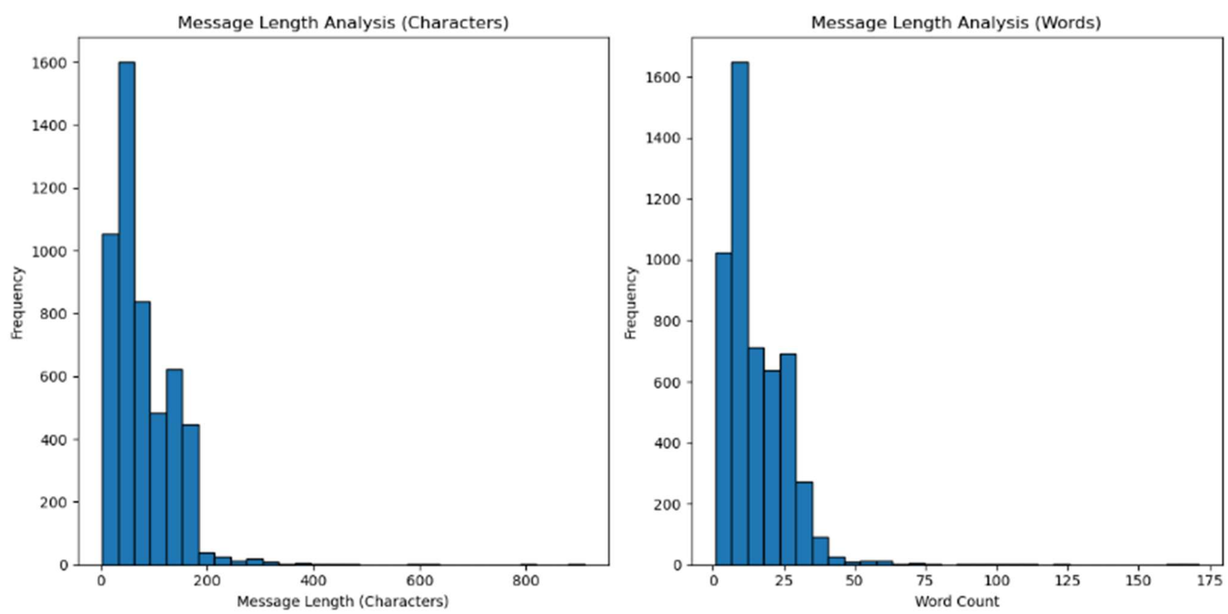


In [117]:

```
# Word Frequency Analysis

import pandas as pd
import nltk
from nltk.corpus import stopwords
from collections import Counter
import matplotlib.pyplot as plt

# Download NLTK stopwords dataset
nltk.download('stopwords')

# Combine all text into a single string
text = ' '.join(df['Message'])

# Tokenize the text
words = nltk.word_tokenize(text)

# Convert to lowercase and remove stopwords
stop_words = set(stopwords.words('english'))
filtered_words = [word.lower() for word in words if word.isalpha() and word.lower() not in stop_words]

# Count word frequencies
word_freq = Counter(filtered_words)

# Get the most common words and their frequencies
most_common_words = word_freq.most_common(20)  # Change the number to get the top N words

# Plot the word frequencies
plt.figure(figsize=(12, 6))
words, frequencies = zip(*most_common_words)
plt.bar(words, frequencies)
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 20 Word Frequencies')
plt.xticks(rotation=45)
plt.show()
```

Message length Analysis:

```
In [118]: # message length Analysis

import pandas as pd
import matplotlib.pyplot as plt

# Calculate message length in characters
df['Message_Length'] = df['Message'].str.len()

# Calculate message length in words
df['Word_Count'] = df['Message'].str.split().apply(len)

# Plot the distribution of message lengths
plt.figure(figsize=(12, 6))

# Histogram of message length in characters
plt.subplot(1, 2, 1)
plt.hist(df['Message_Length'], bins=30, edgecolor='black')
plt.xlabel('Message Length (Characters)')
plt.ylabel('Frequency')
plt.title('Message Length Analysis (Characters)')

# Histogram of message length in words
plt.subplot(1, 2, 2)
plt.hist(df['Word_Count'], bins=30, edgecolor='black')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.title('Message Length Analysis (Words)')

plt.tight_layout()  # Ensures proper spacing of subplots
plt.show()
```



Code Explanation:

Import Libraries: The code begins by importing two important libraries: pandas for data handling and matplotlib.pyplot for creating visual plots and charts.

Load Data: It assumes we have a dataset in a CSV file (replace 'your_data.csv' with our actual file name) and uses pd.read_csv to load this data into a DataFrame called 'df.'

Calculate Message Length:

1. It creates two new columns in the DataFrame: 'Message_Length' and 'Word_Count.'

2. 'Message_Length' stores the number of characters in each text message.

3. 'Word_Count' stores the number of words in each text message.

Plot Distribution:

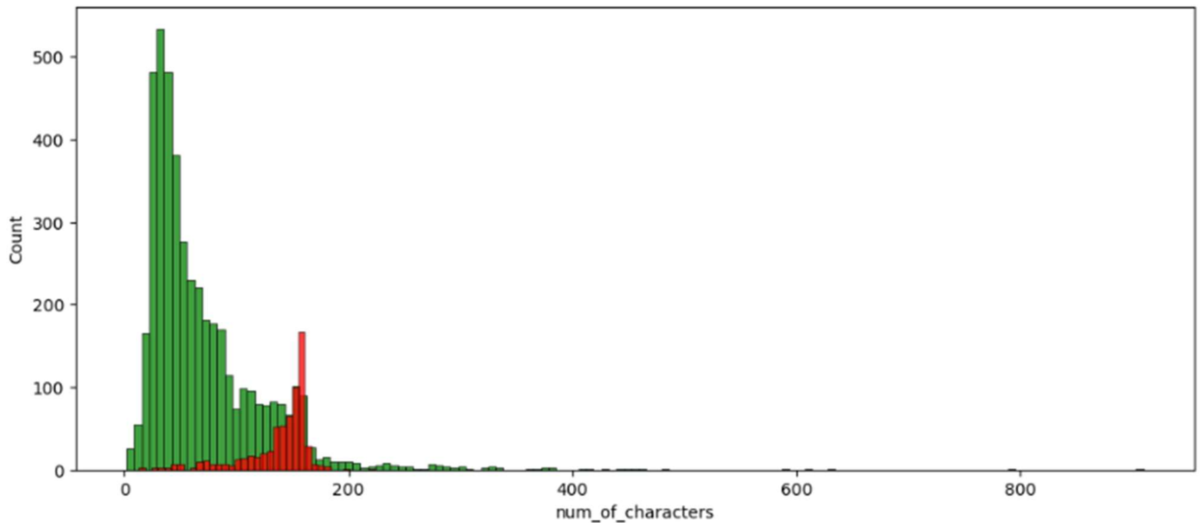
- The code sets up a plot area using plt.figure(figsize=(12, 6)), which is like preparing a canvas for drawing.
- It creates two histograms (bar charts) side by side to analyze the distribution of message lengths.
- The first histogram shows the distribution of message lengths in characters.
- The second histogram shows the distribution of message lengths in words.

Customizing Plots:

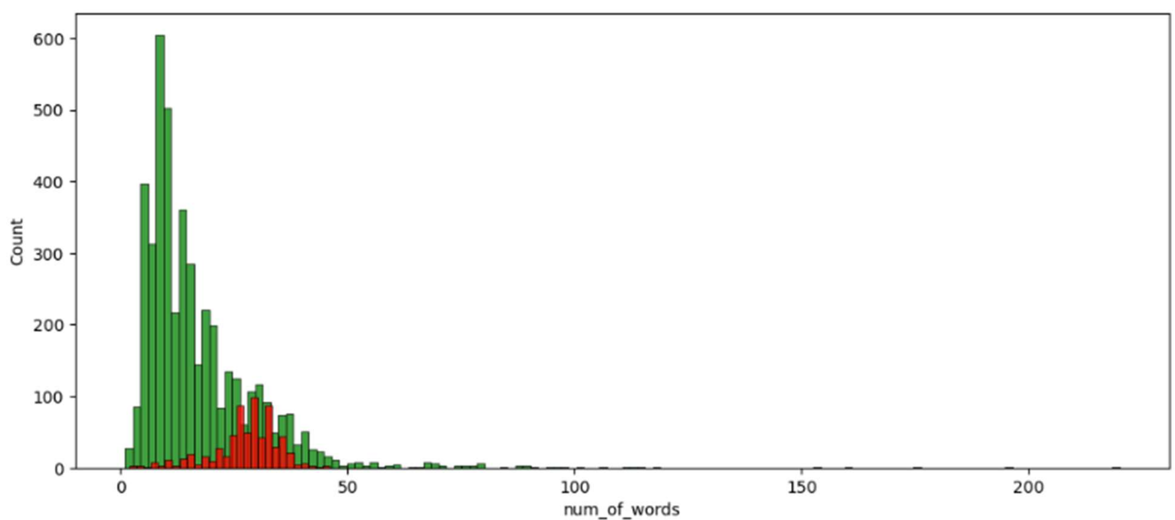
- For each histogram, it specifies the number of bins (divisions) and the color of the edges of the bars to make the plot visually appealing.
- It labels the X and Y axes and gives each histogram a title.
- Show the Plots: plt.tight_layout() ensures that the two plots have proper spacing between them, and plt.show() displays the plots on the screen.

HISTOGRAM REPRESENTATION:

```
In [66]: # Plot a histogram for 'num_of_characters' column

         plt.figure(figsize=(12,5))
         sns.histplot(df[df['Type']==0]['num_of_characters'],color='green')
         sns.histplot(df[df['Type']==1]['num_of_characters'],color='red')

Out[66]: <Axes: xlabel='num_of_characters', ylabel='Count'>
```




```
In [67]: # Plot a histogram for 'num_of_words' column

         plt.figure(figsize=(12,5))
         sns.histplot(df[df['Type']==0]['num_of_words'],color='green')
         sns.histplot(df[df['Type']==1]['num_of_words'],color='red')

Out[67]: <Axes: xlabel='num_of_words', ylabel='Count'>
```



Plot Histogram for 'num_of_characters' column:

- A histogram is created to visualize the distribution of the 'num_of_characters' column in the DataFrame.
- The 'ham' messages are represented in green, and the 'spam' messages are represented in red.

- The first plt.figure(figsize=(12,5)) statement specifies the size of the plot figure for the 'num_of_characters' histogram.

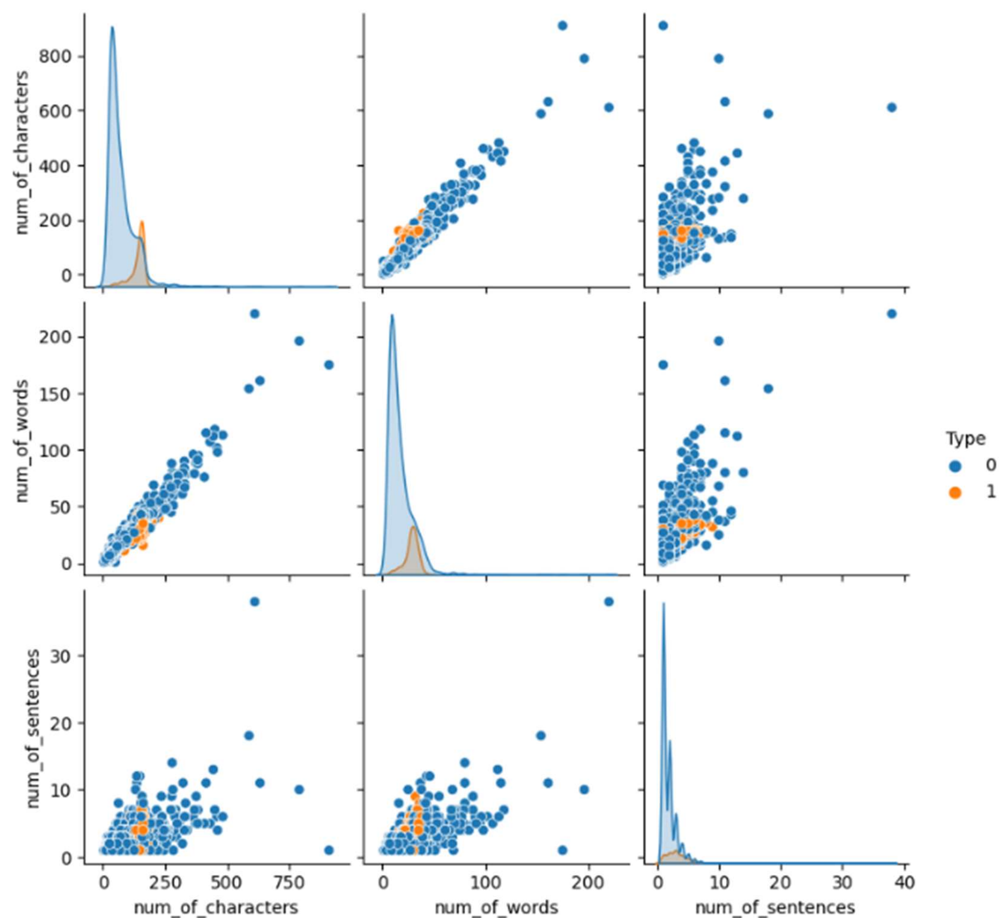
Plot Histogram for 'num_of_words' column:

- Another histogram is plotted, this time for the 'num_of_words' column in the DataFrame.
- Just like before, 'ham' messages are shown in green, and 'spam' messages are in red.
- The second plt.figure(figsize=(12,5)) statement sets the size of the plot figure for the 'num_of_words' histogram.

PAIR PLOT REPRESENTATION:

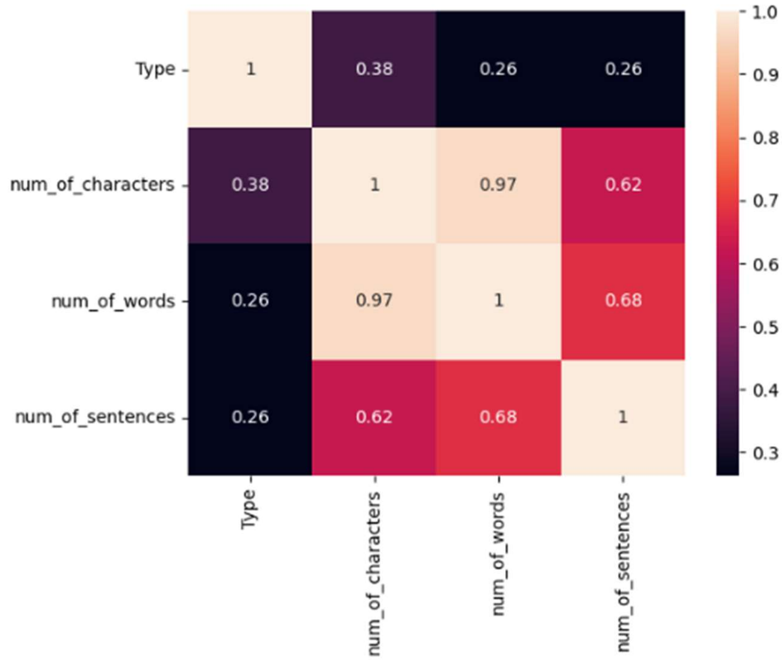
```
In [68]: # Plot a pair plot to visualize the relationships between the variables in the dataset by pairing them in a grid

         sns.pairplot(df,hue='Type')
```

```
Out[68]: <seaborn.axisgrid.PairGrid at 0x11592516910>
```



HEAT MAP:

```
In [69]: # Plot a heat map to visualize the correlation matrix of values
         # Exclude non-numeric columns (e.g., 'text' is a non-numeric column)
         import numpy as np
         numeric_df = df.select_dtypes(include=[np.number])

         # Plot the correlation matrix for numeric columns
         sns.heatmap(numeric_df.corr(), annot=True)

Out[69]: <Axes: >
```



- **Filter Numeric Columns:**

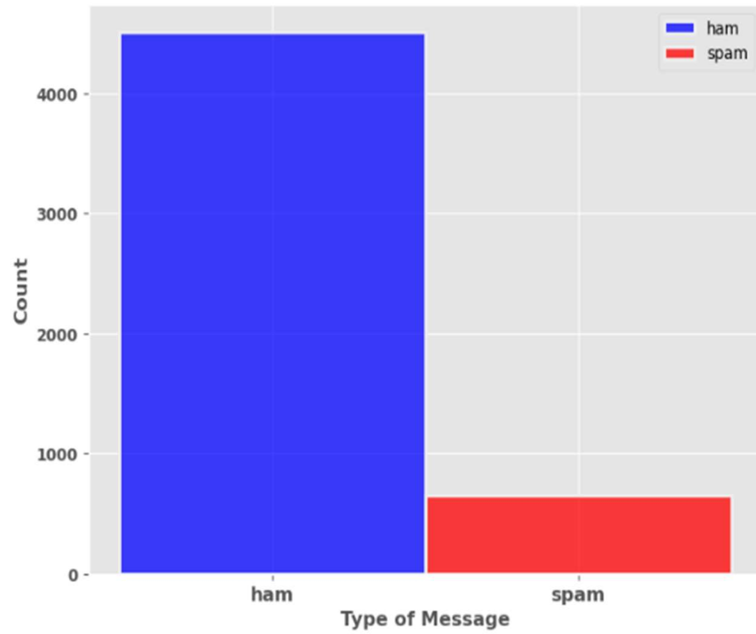
    The code starts by selecting the numeric columns from the DataFrame df and creates a new DataFrame called numeric_df. Non-numeric columns (e.g., 'text') are excluded.

- **Plot a Heat Map:**
    - ✓ A heat map is generated to visually represent the correlation matrix of values in the numeric_df.
    - ✓ The sns.heatmap function from the Seaborn library is used for this purpose.
    - ✓ Correlation values are annotated and displayed within the heatmap.

DATA VISUALIZATION:

Histogram to show the distribution of two message types, "ham" and "spam," using Matplotlib and Seaborn.

```
In [111]: # creating a histogram to visualize the distribution of two types of messages, "ham" and "spam".
from matplotlib import style
%matplotlib inline
plt.style.use('ggplot')
plt.figure(figsize=(8,6))
sns.histplot(df[df['Type']==0]['Type'],color='blue', label='ham', linewidth=2)
sns.histplot(df[df['Type']==1]['Type'],color='red', label='spam', linewidth=2)
plt.xticks([0, 1], ['ham', 'spam'],fontsize=12,fontweight='bold')
plt.yticks(fontweight='bold')
plt.xlabel('Type of Message',fontweight='bold')
plt.ylabel('Count',fontweight='bold')
plt.legend()
plt.show()
```



- **Import Libraries:**

  The code imports necessary libraries for plotting.

- **Set Plot Style:**

  It defines the plot style as 'ggplot' for a specific appearance.

- **Create a Figure:**

  A figure of size 8x6 inches is prepared for the plot.

- **Plot Histograms:**

  Two histograms are plotted, one for "ham" messages in blue and one for "spam" messages in red.

- **Set Axis Labels and Ticks:**

Labels and ticks for the x and y axes are configured. The x-axis shows "ham" and "spam," and the font is made bold.

- **Set Axis Titles:**

  Titles for the x and y axes are added.

- **Add Legend:**

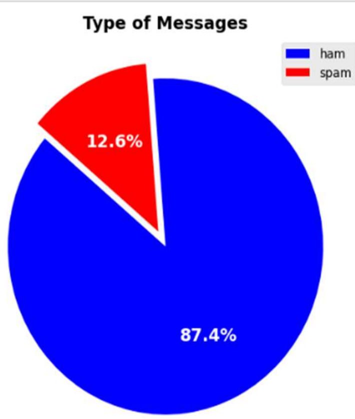
  A legend is included to distinguish "ham" and "spam."

- **Display the Plot:**

  The plot is shown in the Jupyter Notebook.

The Below code creates a pie chart to illustrate the distribution of two types of messages, "ham" and "spam," using Matplotlib.

```
# creating a pie chart to visualize the distribution of two types of messages, "ham" and "spam"
ham_count = len(df[df['Type'] == 0])
spam_count = len(df[df['Type'] == 1])
labels = ['ham', 'spam']
sizes = [ham_count, spam_count]
colors = ['blue', 'red']
explode = (0, 0.1)
plt.figure(figsize=(8, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',textprops={'color': 'white', 'size':'14','weight
plt.title('Type of Messages', fontsize=14, fontweight='bold')
plt.legend()
plt.show()
```



- **Calculate Message Counts:**

  The code calculates the number of "ham" and "spam" messages in the dataset and stores them in ham_count and spam_count variables.

- **Prepare Data:**

  It prepares the data for the pie chart, specifying labels, sizes, colors, and an "explode" effect (to separate one slice).

- **Create a Figure:**

  A figure of size 8x6 inches is set up for the pie chart.

- **Plot the Pie Chart:**

  A pie chart is generated using the prepared data. The explode effect separates the "spam" slice slightly. Labels and percentages are displayed inside the chart with white text and bold font style.

- **Set Title:**

  A title, "Type of Messages," is added to the chart with a bold font.

- **Add Legend:**

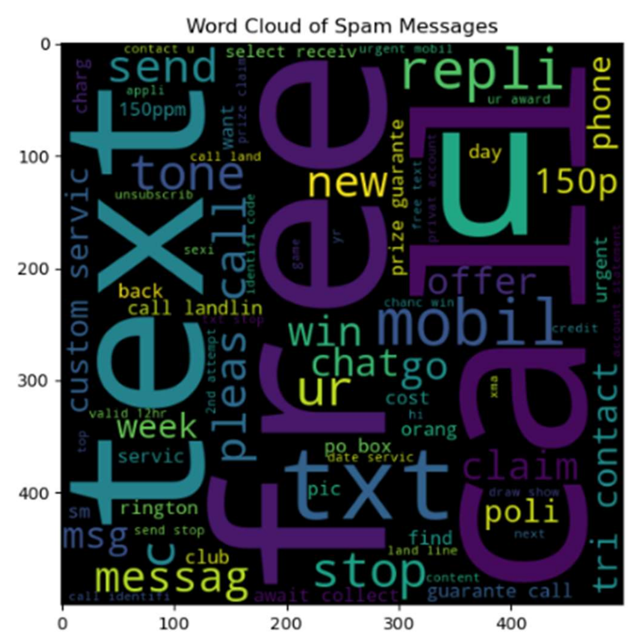
  A legend is included to identify "ham" and "spam" slices.

- **Display the Pie Chart:**

  The pie chart is displayed for visualization.

The Below code generates a word cloud to visualize the most frequent words in spam messages.

```
In [113]: # Word cloud of spam messages

          wc=WordCloud(width=500,height=500,min_font_size=10,background_color='black')
          spam_wc=wc.generate(df[df['Type'] == 1]['transformed_text'].str.cat(sep=" "))
          plt.rcdefaults()
          plt.figure(figsize=(15,6))
          plt.title("Word Cloud of Spam Messages")
          plt.imshow(spam_wc)

Out[113]: <matplotlib.image.AxesImage at 0x115b7a4c3d0>
```



- **Initialize WordCloud:**

  The code sets up a WordCloud object with specific properties, such as the size of the word cloud image, minimum font size, and the background color (black).

- **Generate Word Cloud for Spam Messages:**

  It creates a word cloud for spam messages by combining the transformed text from these messages into a single text block. This step forms a word cloud based on the most common words found in spam messages.

- **Reset Plot Defaults:**

  This line resets any previous plot settings to default values.

- **Create a Figure:**

  A figure for displaying the word cloud is defined with a size of 15x6 units.

- **Set Title:**

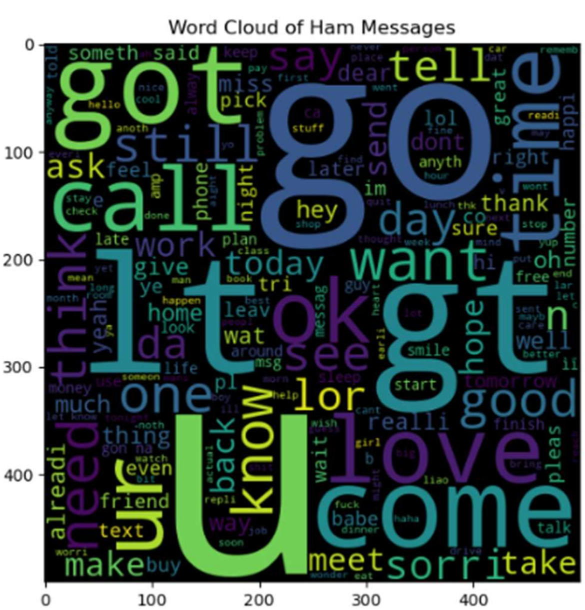
  The title "Word Cloud of Spam Messages" is added to the figure.

  The title "Word Cloud of Ham Messages" is added to the figure.

- **Display the Word Cloud:**

  The word cloud is displayed within the figure, showcasing the most prominent words in spam messages by their size and frequency.

```
In [114]: # Word cloud of ham messages

          ham_wc=wc.generate(df[df['Type'] == 0]['transformed_text'].str.cat(sep=" "))
          plt.rcdefaults()
          plt.figure(figsize=(15,6))
          plt.title("Word Cloud of Ham Messages")
          plt.imshow(spam_wc)

Out[114]: <matplotlib.image.AxesImage at 0x11593c40e50>
```



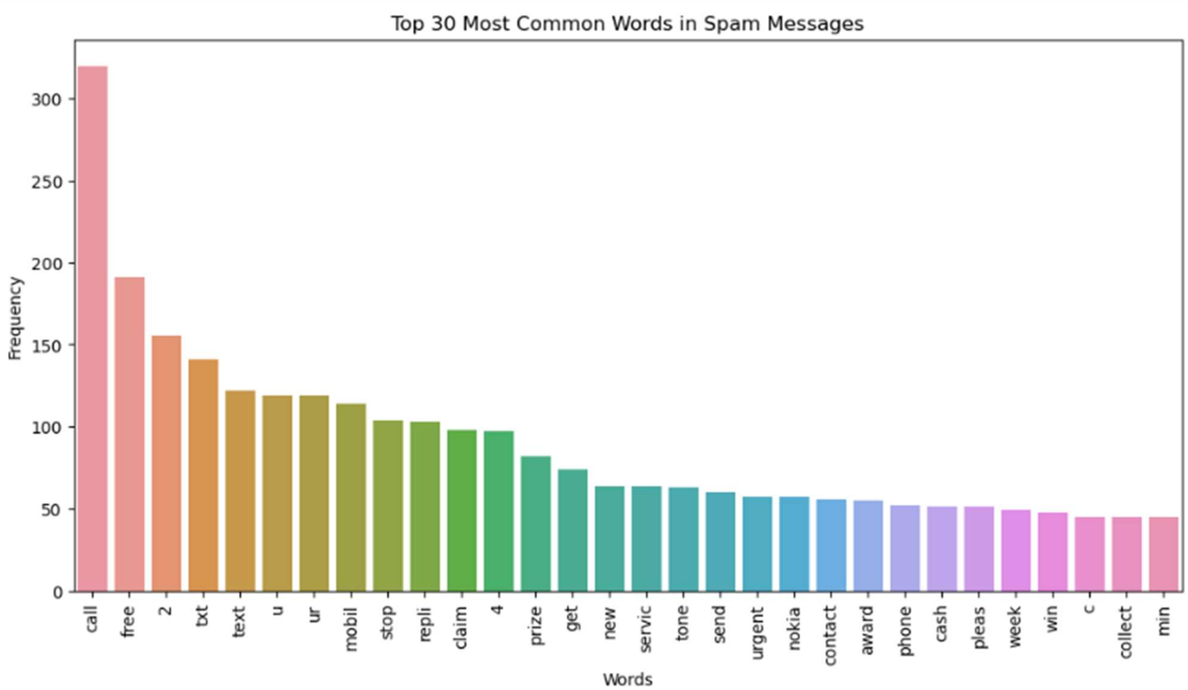
MOST USED WORDS IN SPAM MESSAGES:

```
In [75]: # Most used words in spam messages

         spam_corpus = []
         for msg in df[df['Type'] == 1]['transformed_text'].tolist():
             for word in msg.split():
                 spam_corpus.append(word)
```

```
In [76]: # number of most used words

         len(spam_corpus)

Out[76]: 9939
```

```
In [77]: spam_corpus

Out[77]: ['free',
          'entri',
          '2',
          'wkli',
          'comp',
          'win',
          'fa',
          'cup',
          'final',
          'tkt',
          '21st',
          'may',
          'text',
          'fa',
          '87121',
          'receiv',
          'entri',
          'question',
          'std',
```

```
In [78]: # Top 30 Most Common Words in Spam Messages
         word_counts = Counter(spam_corpus)
         common_words = dict(word_counts.most_common(30))
         words = list(common_words.keys())
         counts = list(common_words.values())

         plt.figure(figsize=(12, 6))
         sns.barplot(x=words, y=counts)
         plt.xticks(rotation='vertical')
         plt.xlabel('Words')
         plt.ylabel('Frequency')
         plt.title('Top 30 Most Common Words in Spam Messages')
         plt.show()
```



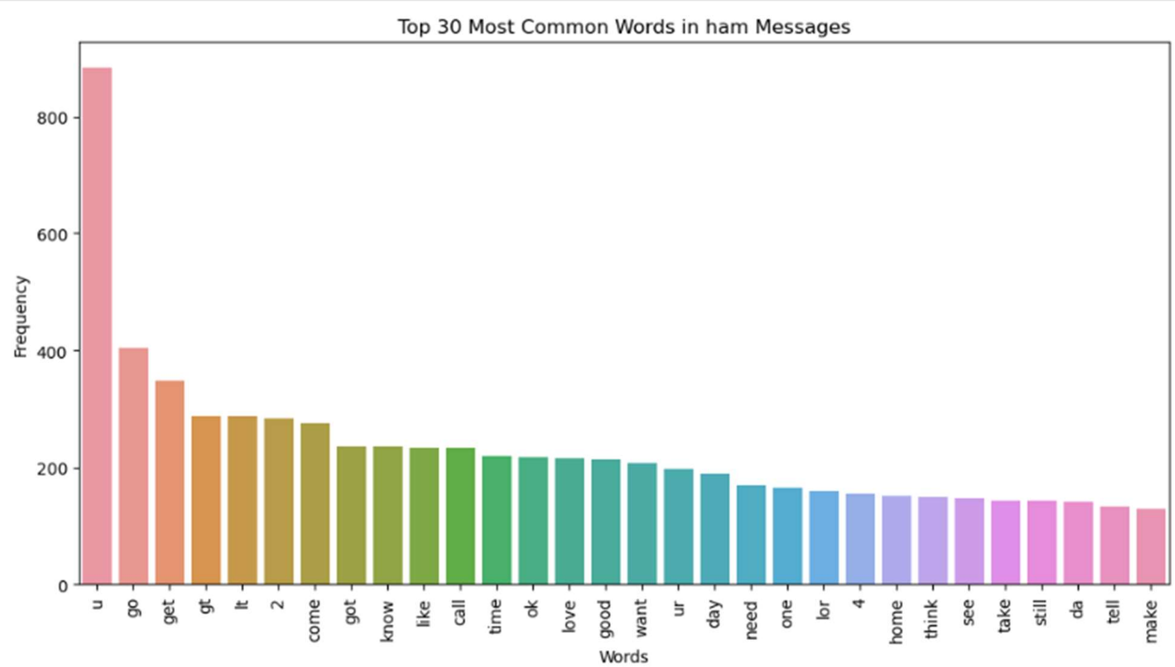
MOST USED WORDS IN HAM MESSAGES:

```
In [79]: # Most used words in ham messages

         ham_corpus = []
         for msg in df[df['Type'] == 0]['transformed_text'].tolist():
             for word in msg.split():
                 ham_corpus.append(word)
```

```
In [80]: # number of most used words

         len(ham_corpus)

Out[80]: 35404
```

```
In [81]: ham_corpus

Out[81]: ['go',
          'jurong',
          'point',
          'crazi',
          'avail',
          'bugi',
          'n',
          'great',
          'world',
          'la',
          'e',
          'buffet',
          'cine',
          'got',
          'amor',
          'wat',
          'ok',
          'lar',
          'joke',
```

```
In [82]: # Top 30 Most Common Words in ham Messages
         word_counts = Counter(ham_corpus)
         common_words = dict(word_counts.most_common(30))
         words = list(common_words.keys())
         counts = list(common_words.values())

         plt.figure(figsize=(12, 6))
         sns.barplot(x=words, y=counts)
         plt.xticks(rotation='vertical')
         plt.xlabel('Words')
         plt.ylabel('Frequency')
         plt.title('Top 30 Most Common Words in ham Messages')
         plt.show()
```



FEATURE EXTRACTION:

Feature extraction is a dimensionality reduction technique used in machine learning and data analysis. It involves transforming or selecting the most relevant information (features) from a dataset while discarding less important or redundant data. Feature extraction is a crucial step in the data preprocessing pipeline and is important for several reasons:

Dimensionality Reduction: Many datasets have a large number of features, which can lead to the curse of dimensionality. This can result in increased computation time, overfitting, and reduced model performance. Feature extraction helps reduce the number of features while retaining essential information.

Improved Model Performance: Feature extraction can lead to better model performance. By selecting the most informative features, models can focus on the relevant patterns in the data, resulting in more accurate predictions and generalization to new data.

Reduced Overfitting: Feature extraction reduces the risk of overfitting, where a model becomes too specialized to the training data and fails to generalize to new data. By reducing the number of features and focusing on the most relevant ones, overfitting can be mitigated.

Enhanced Interpretability: Simplifying the dataset through feature extraction can make it easier to interpret the model's results. It becomes more straightforward to understand the factors influencing the model's decisions.

Computational Efficiency: Feature extraction reduces the computational load, as models process a smaller set of features, leading to faster training and inference times.

Noise Reduction: Some features in a dataset may contain noise or be irrelevant to the task at hand. Feature extraction can help filter out these noisy features, improving the quality of the data used for modeling.

Improved Data Visualization: Feature extraction can make data visualization more effective. By reducing the data to a manageable number of dimensions, it becomes easier to create meaningful visualizations.

Domain-Specific Knowledge Integration: Domain experts can often guide feature extraction by selecting or engineering features that are known to be relevant for a particular problem. This can lead to more effective models.

Enhanced Data Preprocessing: Feature extraction is a critical component of data preprocessing, which is essential for preparing data for machine learning. It helps clean and refine the data before feeding it to models.

```
In [84]: # Import necessary dependencies for feature extraction

         from sklearn.feature_extraction.text import TfidfVectorizer

In [85]: # Perform feature extraction

         tfidf=TfidfVectorizer()

         # x is the input feature
         x=tfidf.fit_transform(df['transformed_text']).toarray()

         # y is the ouput feature
         y=df['Type'].values

In [86]: # Input feature

         x

Out[86]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])

In [87]: x.shape

Out[87]: (5169, 6708)

In [88]: # Output feature

         y

Out[88]: array([0, 0, 1, ..., 0, 0, 0])

In [89]: y.shape

Out[89]: (5169,)
```

CODE EXPLANATION:

Import Dependencies:

- The code imports the necessary dependencies, specifically the TfidfVectorizer from scikit-learn (sklearn), which is used for feature extraction from text data.

Initialize TF-IDF Vectorizer:

- It creates a TF-IDF vectorizer object called tfidf. TF-IDF stands for Term Frequency-Inverse Document Frequency, a method for converting text data into numerical feature vectors.

Perform Feature Extraction:

- The code applies feature extraction to the 'transformed_text' column of the DataFrame df using the TF-IDF vectorizer.
- The result is stored in variable x, representing the input features.

Input Feature (x) Explanation:

- The x variable contains the transformed text data as numerical feature vectors after TF-IDF feature extraction.
- x.shape is used to determine the shape of the x array, which reveals the number of samples and features.

Output Feature (y) Preparation:

- The code creates the output feature y by extracting the 'Type' column from the DataFrame, which likely represents the classification labels.

Output Feature (y) Explanation:

- The y variable contains the output feature, which typically represents the class labels (e.g., 'ham' or 'spam').
- y.shape is used to determine the shape of the y array, revealing the number of samples.

BUILDING MODEL FOR SPAM CLASSIFIER:

```
In [90]: # import the dependency for spliting train and test sets
         from sklearn.model_selection import train_test_split

In [91]: # Split the input and output features into train and test sets
         x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=2)
```

Import Dependency for Train-Test Split:

- The code imports the necessary dependency, train_test_split, from scikit-learn (sklearn). This function is used to split the dataset into training and testing subsets.

Split the Data into Train and Test Sets:

- The code performs the actual data split.
- x represents the input features, and y represents the output features or class labels.
- The data is split into training and testing sets using train_test_split.
- x_train and y_train represent the training input and output features, respectively.
- x_test and y_test represent the testing input and output features, respectively.
- The test_size parameter specifies the proportion of data to allocate for the test set (in this case, 20% of the data).
- The random_state parameter ensures reproducibility by setting a specific random seed.

```
In [1]: !pip install tensorflow
```

```
In [95]: # Import the necessary dependencies for creating a neural network

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
```

```
In [96]: # Create a neural network with a input layer, three hidden layers and a ouput layer
# No.of neurons in input layer is determined by input shape and input layer uses 'relu' activation function
# 500 neurons in 1st hidden layer and it uses 'relu' activation function
# 500 neurons in 2nd hidden layer and it uses 'relu' activation function
# 100 neurons in 3rd hidden layer and it uses 'relu' activation function
# 100 neurons in 4th hidden layer and it uses 'relu' activation function
# 1 neuron in output  layer and it uses 'sigmoid' activation function

model=Sequential()
model.add(Dense(units=500,input_shape=(6708,),activation='relu'))
model.add(Dense(units=500,activation='relu'))
model.add(Dense(units=100,activation='relu'))
model.add(Dense(units=100,activation='relu'))
model.add(Dense(units=1,activation='sigmoid'))
```

```
In [98]: # Compile the model
# Use binary_crossentropy loss function
# Use adam optimizer
# Use accuracy metrics for monitoring

model.compile(loss='binary_crossentropy' ,optimizer='adam',metrics='accuracy')
```

```
In [99]: # Train the model on the  dataset

model.fit(x_train,y_train,epochs=10,batch_size=64)

Epoch 1/10
65/65 [==============================] - 7s 52ms/step - loss: 0.2721 - accuracy: 0.8672
Epoch 2/10
65/65 [==============================] - 3s 49ms/step - loss: 0.0975 - accuracy: 0.9775
Epoch 3/10
65/65 [==============================] - 3s 49ms/step - loss: 0.0139 - accuracy: 0.9981
Epoch 4/10
65/65 [==============================] - 3s 48ms/step - loss: 7.9564e-04 - accuracy: 0.9998
Epoch 5/10
65/65 [==============================] - 3s 49ms/step - loss: 1.5202e-04 - accuracy: 1.0000
Epoch 6/10
65/65 [==============================] - 3s 47ms/step - loss: 6.1699e-05 - accuracy: 1.0000
```

Install TensorFlow:

The code begins by installing the TensorFlow library using the !pip install tensorflow command. TensorFlow is a deep learning framework.

Import Dependencies for Neural Network:

It imports the necessary dependencies for creating a neural network using TensorFlow and some functions for model evaluation from scikit-learn.

Create a Neural Network:

- A neural network is defined with the following architecture:
  - ✓ Input layer with a ReLU activation function.
  - ✓ Three hidden layers with 500, 500, and 100 neurons, each using ReLU activation.
  - ✓ An output layer with 1 neuron using a sigmoid activation function.

Compile the Model:

- The model is compiled with the following settings:
  - ✓ Loss function: Binary cross-entropy (commonly used for binary classification tasks).
  - ✓ Optimizer: Adam optimizer (a popular optimization algorithm).
  - ✓ Metric for monitoring: Accuracy.

Train the Model:

The model is trained on the training data (x_train and y_train) for 10 epochs (training iterations), with a batch size of 64.

Model Evaluation:

- ✓ The model's performance is evaluated on the test data (x_test and y_test).
- ✓ The evaluation results are stored in the results variable.
- ✓ The code then prints the model's accuracy, displaying it as a percentage.

```
In [120]: # Model evaluation

          results = model.evaluate(x_test, y_test)
          print('Accuracy: {:.2%}'.format(results[1]))

          33/33 [==============================] - 1s 5ms/step - loss: 0.2179 - accuracy: 0.9729
          Accuracy: 97.29%
```

Model:

```
In [121]: # Use the model on user input

def transform_text(text):
    # Convert the message to lower case
    text=text.lower()

    # Tokenize the message
    text=nltk.word_tokenize(text)

    # Remove the special characters in the message
    y=[]
    for i in text:
        if i.isalnum():
            y.append(i)
    text=y[:]
    y.clear()

    # Remove the stop words and punctuations in the message
    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    text=y[:]
    y.clear()

    # Stemming
    for i in text:
        y.append(ps.stem(i))

    return " ".join(y)

user_text = input('Input the text: ')
preprocessed_text=transform_text(user_text)
vectorized_text=tfidf.transform([preprocessed_text]).toarray()

prediction = model.predict(vectorized_text)

print('Spam level: {:.2%}'.format(prediction[0][0]))

if prediction > 0.8:
    print('Spam!')
else:
    print('Not spam!')
```

Input the Message:

```
Input the text: freemsg hey there darling it s been 3 week s now
```

Output:

```
Input the text: freemsg hey there darling it s been 3 week s now and no word back  i d like some fun you up for it still  tb ok
xxx std chgs to send    1 50 to rcv,freemsg hey darl 3 week word back like fun still tb ok xxx std chg send 1 50 rcv
1/1 [==============================] - 0s 33ms/step
Spam level: 100.00%
Spam!
```

CODE EXPLANATION:

Text Preprocessing (transform_text function):

- ✓ The transform_text function takes a user's input text as its argument.
- ✓ It starts by converting the input text to lowercase to ensure consistent handling of text.
- ✓ The text is tokenized, meaning it's split into individual words or tokens using the NLTK library's word_tokenize function.
- ✓ Special characters in the text are removed, and the remaining words are collected in a list called y. This step helps clean the text.
- ✓ Stop words (common words like "the," "and," "is") and punctuation are removed from the text to keep only meaningful words.
- ✓ The text goes through stemming using the NLTK Porter Stemmer (ps.stem(i)) to reduce words to their root form.
- ✓ Finally, the processed words are joined to create a clean, preprocessed text.

User Input:

- ✓ The code prompts the user to input text: user_text.

Preprocess User Input:

- ✓ The transform_text function is called to preprocess the user's input text (user_text), making it suitable for model input.
- ✓ The preprocessed text is stored in the variable preprocessed_text.

Vectorize the Text:

- ✓ The preprocessed text is transformed into numerical feature vectors using the TF-IDF vectorizer (tfidf.transform). This step converts the text into a format the model can understand.
- ✓ The resulting feature vector is stored in the variable vectorized_text.

Make a Prediction:

- ✓ The model is used to predict whether the text is spam or not. The prediction is based on the provided feature vector (vectorized_text).
- ✓ The prediction result is stored in the variable prediction.

Display the Result:

- ✓ The code displays the "Spam level," which is the model's prediction score as a percentage.
- ✓ If the prediction score is greater than 80% (0.8), it's classified as "Spam!"; otherwise, it's classified as "Not spam!"