BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER

Phase-3 Submission Document

Project Title: AI-Powered Spam Classifier

Phase-3: Development Part 1

Topic: Start building the smarter AI-Powered spam classifier by loading and pre-processing the dataset.

Team Members:

- R.DHARSHINI(team leader)
- M.YAMINI
- S.MINITHRA
- L.SHIBIYA CHRISTY
- K.MUTHUKAMATCHI

SPAM CLASSIFICATION

INTRODUCTION:

In the digital age, where communication and information sharing occur primarily through emails, the problem of spam has become increasingly pervasive and burdensome. Spam emails clutter inboxes, often containing malicious content or deceptive schemes. To combat this issue, machine learning and data science techniques come to the forefront, empowering us to create robust spam email classifiers. Loading and preprocessing a spam dataset is the initial and indispensable stage in this endeavor.

Loading a spam dataset involves accessing and bringing the data into a suitable format for analysis. Preprocessing, on the other hand, entails a series of data

transformations, cleaning, and feature engineering steps to prepare the dataset for machine learning algorithms. This process ensures that the data is in a form that can be readily consumed by the models, ultimately leading to more accurate and efficient spam classification.

Loading and preprocessing a spam dataset is a critical step in the field of machine learning and data science, particularly when working on spam email classification or related tasks. Spam datasets typically consist of a large collection of emails, some of which are spam (unsolicited and often unwanted messages) and others that are legitimate, or "ham." To effectively build a spam email classifier or perform any meaningful analysis, we need to follow a systematic approach for loading and preprocessing the dataset.

GIVEN DATASET:

DATASET LINK:

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

NECESSARY STEPS TO FOLLOW:

1. **IMPORT LIBRARIES:**

Start by importing the necessary libraries.

PROGRAM:

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.pipeline import Pipeline
from sklearn.externals import joblib
```

2. LOAD DATASET:

To load a dataset for a spam classifier, you'll typically be working with a labeled dataset where each item is categorized as either spam or not spam (ham).We can use Pandas to load the dataset from the CSV file.

PROGRAM:

```
df = pd.read_csv("/kaggle/input/sms-spam-collection-dataset/spam.csv",
encoding='ISO-8859-1')
pd.read( )
```

3. EXPLORATORY DATA ANALYSIS(EDA):

Exploratory Data Analysis (EDA) is an essential step in understanding your dataset before building a spam classifier. It helps you gain insights into the data's characteristics, identify patterns, and determine the most relevant features.

PROGRAM:

```
# Visualize the distribution of spam and non-spam emails
sns.countplot(data['label'])
plt.xlabel('Label')
plt.ylabel('Count')
plt.title('Distribution of Spam and Non-Spam Emails')
plt.show()

#Word cloud for spam emails to visualize common words
spam_text = " ".join(data[data['label'] == 1]['text'])
wordcloud = WordCloud(width=800, height=400).generate(spam_text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud for Spam Emails")
plt.show()

# Text length analysis
data['text_length'] = data['text'].apply(len)
sns.histplot(data, x='text_length', hue='label', kde=True)
plt.xlabel('Text Length')
plt.title('Text Length Distribution by Label')
```

```
plt.show()

# display the plots as desired
plt.show()
```

4. FEATURE ENGINEERING:

Feature engineering is a crucial step in building an AI-powered spam classifier. It involves creating new features or transforming existing ones to improve the performance of the classifier.

PROGRAM:

```
# Feature Engineering (TF-IDF vectorization)
vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(data['text'])
y = data['target']
```

5. SPLIT THE DATA:

In a spam classifier, you typically split your dataset into two or three parts: a training set, a validation set, and a test set. The purpose of this split is to train your model, tune its parameters, and evaluate its performance.

PROGRAM:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=42)
```

6. FEATURE SCALING:

Feature scaling is an essential preprocessing step in many machine learning algorithms, including those used for spam classification. Feature scaling helps ensure that all your features (attributes or variables) have a similar scale, which can improve the performance of various machine learning models and algorithms.

PROGRAM:

```
# Feature scaling (e.g., standardization) for non-text features
# Assuming you have numerical or other non-text features
scaler = StandardScaler()
X_train_non_text = scaler.fit_transform(data[['feature1', 'feature2']])
X_test_non_text = scaler.transform(data[['feature1', 'feature2']])
```

IMPORTANCE OF LOADING AND PRE - PROCESSING DATASET:

loading and preprocessing your dataset sets the foundation for building a smarter AI-powered spam classifier. It ensures that your model is trained on high-quality, relevant data, and is capable of making accurate predictions and generalizing to new, unseen emails. Additionally, it helps you address issues like data imbalance, outliers, and data quality that can significantly impact the performance of your spam classifier.

Proper data loading and preprocessing are crucial for obtaining meaningful insights and building accurate classifiers.

CHALLENGES INVOLVED IN LOADING AND PRE - PROCESSING THE DATASET IN AI - POWERED SPAM CLASSIFIER:

1. **DATA CLEANING:**

   Many datasets contain noisy and unstructured data, which requires extensive cleaning. This includes removing HTML tags, special characters, and irrelevant content from email text.

2. **DATA PRIVACY:**

   Email datasets can contain sensitive information. Ensuring data privacy and compliance with regulations such as GDPR can be a challenge.

3. **LABEL NOISE:**

   Labeling data as spam or non-spam can contain errors. Handling label noise and ensuring the accuracy of training data is a challenge.

4. **UPDATING MODELS:**

   Spam classifiers need to be regularly updated to adapt to new spamming techniques. Implementing an effective update mechanism is a challenge.

5. **HANDLING THE HTML AND RICH TEXT:**

Email messages often contain HTML and rich text, which require special handling to extract relevant text data. Parsing and cleaning HTML can be challenging.

STEPS TO OVERCOME THE CHALLENGES IN LOADING AND PRE - PROCESSING THE DATASET IN AI - POWERED SPAM CLASSIFIER:

1. **DATA CLEANING:**

Implement robust data cleaning procedures to remove HTML tags, special characters, and irrelevant content from email text.

Use regular expressions and natural language processing (NLP) techniques to identify and handle messy data.

2. **DATA PRIVACY:**

Anonymize or pseudonymize sensitive data to protect user privacy.
Ensure compliance with data protection regulations (e.g., GDPR).

3. **DATA LABELING:**

Automate labeling where possible, but also perform manual verification and validation of labels. Ensure consistency and accuracy in the labeling process.

4. UPDATING MODELS:

When you're building an AI-powered spam classifier, it's essential to keep your model up to date to ensure that it continues to perform well as spam and email patterns evolve over time.

5. HANDLING HTML AND RICH TEXT:

Remove HTML tags from the text. This can be done using libraries like Beautiful Soup or regular expressions.
Convert all text to lowercase to ensure case insensitivity.
Normalize whitespace, including extra spaces, line breaks, and tabs.

1. LOADING THE DATASET:

Loading a dataset for an AI-powered spam classifier is a crucial initial step in the development of your spam classification model. The dataset typically contains labeled examples of spam and non-spam (ham) messages, which are used to train and evaluate the performance of your AI model.

1.CHOOSE A DATASET:

Select a suitable dataset for spam classification. Several publicly available email spam datasets, such as the Enron Spam Dataset, TREC Spam Corpus, and SpamAssassin Public Corpus, can be used. You can also create your own dataset by collecting and labeling spam and non-spam emails.

2. DATA PREPARATION:

Prepare your dataset in a structured format. Ensure that it contains two main components: the email text (message content) and the corresponding labels (spam or non-spam).
It's essential to have a clear labeling convention, such as using binary labels (1 for spam, 0 for non-spam) or text labels (e.g., 'spam' and 'ham').

3.DATA LOADING:

Import the necessary libraries for data manipulation and machine learning, such as pandas for data handling.
Load your dataset into a data structure, such as a Pandas DataFrame, from the source file, which could be a CSV, TSV, or other structured format.

4. DATA EXPLORATION:

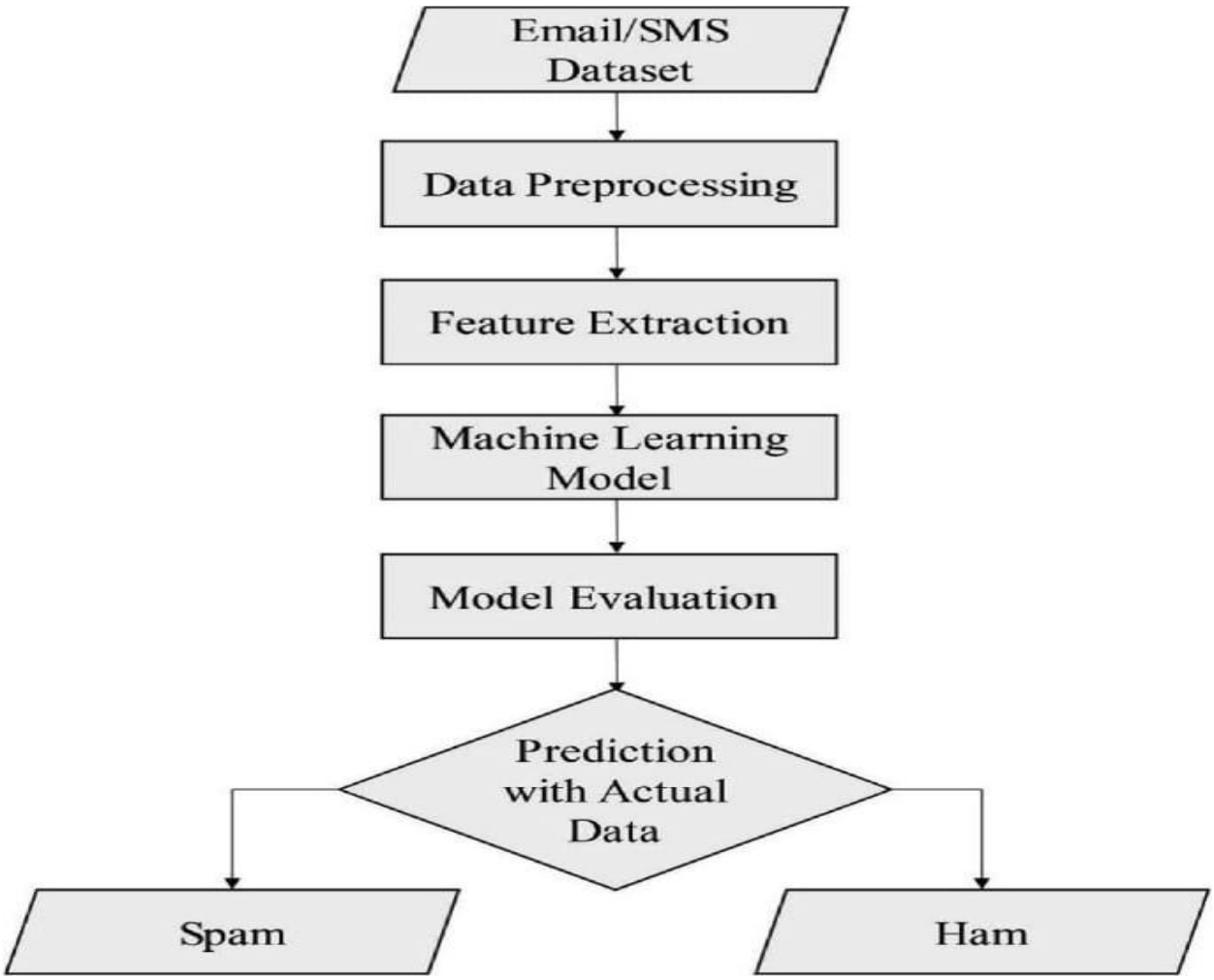
Explore your dataset to get a sense of its structure and content. Use methods like head( ),describe ( ),info( ) to understand the data.

5. DATA PREPROCESSING:

Clean and preprocess the data. Common preprocessing steps may include

1. **Handling missing values, if any.**
2. **Text normalization (e.g., converting to lowercase).**
3. **Tokenization to break text into words or phrases.**
4. **Removing stop words and punctuation.**

5. **Handling HTML tags and special characters (if the dataset contains HTML content).**



PROGRAM:

```
#IMPORTING LIBRARIES

import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import os
import string
import keras
import nltk
import random
import plotly.express as px
import plotly.figure_factory as ff
import spacy

from plotly import graph_objs as go
from PIL import Image
from wordcloud import WordCloud , STOPWORDS , ImageColorGenerator
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

from tqdm import tqdm
from collections import Counter , defaultdict
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.callbacks import ModelCheckpoint  , ReduceLROnPlateau
from keras.optimizers import Adam
from keras.models import Sequential
from keras.initializers import Constant
from keras.layers import Dense , LSTM ,Embedding , BatchNormalization
,Dropout ,Bidirectional , Flatten ,GlobalMaxPool1D
from sklearn.metrics import confusion_matrix , classification_report,

accuracy_score ,f1_score

#LOAD DATA
```

```
data_path = '/kaggle/input/sms-spam-collection-dataset/spam.csv'
data =  pd.read_csv(data_path , encoding = 'latin')
data = data.drop(['Unnamed: 2' ,'Unnamed: 3' , 'Unnamed: 4'] , axis=1)
data.columns = ['Target', 'Message']
data.reset_index()
data.head()
```

OUTPUT:

	Target	Messages
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...

#EXPLORATORY DATA ANALYSIS

```
data['message_length'] = data['Message'].apply(lambda x: len(x.split(" ")))
data.head()
```

OUTPUT:

	Target	Message	Message length
0	ham	Go until jurong point, crazy.. Available only ...	20
1	ham	Ok lar... Joking wif u oni...	6
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	28
3	ham	U dun say so early hor... U c already then say...	11
4	ham	Nah I don't think he goes to usf, he lives aro...	13

VISUALIZING THE DATASET:

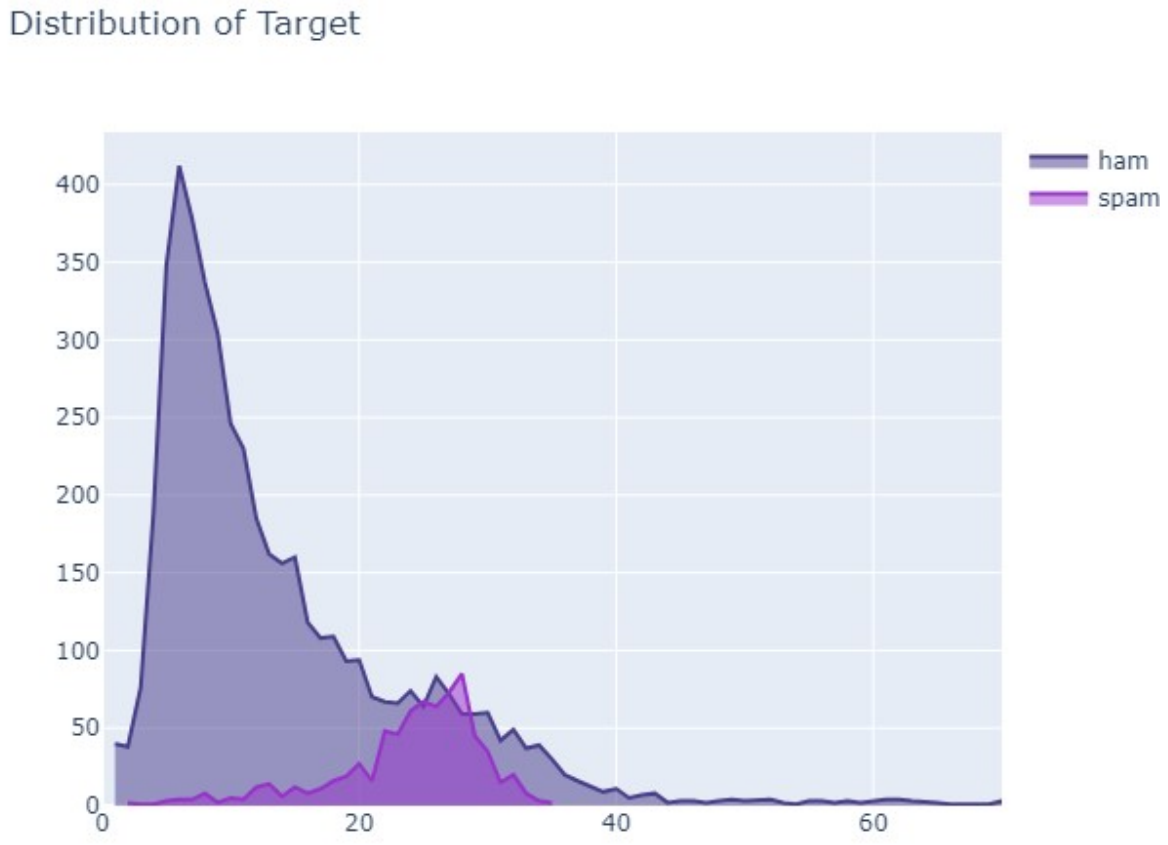
PROGRAM:

```
data['Target'].value_counts()
Ham_len=
```

```
data[data['Target']=='ham']['message_length'].value_counts().sort_index()
Spam_len=
 data[data['Target']=='spam']['message_length'].value_counts().sort_index()
 fig = go.Figure()
 fig.add_trace(go.Scatter(
 x = Ham_len.index ,
 y = Ham_len.values ,
 name= 'ham' ,
 fill= 'tozeroy',
 marker_color = 'darkslateblue',
  ))
 fig.add_trace(go.Scatter(
 x = Spam_len.index ,
 y = Spam_len.values ,
 name = 'spam' ,
 fill = 'tozeroy',
 marker_color = 'darkorchid' ,
 ))
 fig.update_layout( title = 'Distribution of Target')
 fig.update_xaxes(range =[0,70])
  fig.show()
```

OUTPUT:

```
Target
ham      4825
spam      747
Name: count, dtype: int64
```



2. PRE - PROCESSING THE DATASET:

Preprocessing the dataset is a crucial step in building an AI-powered spam classifier. Data preprocessing helps clean and transform the raw dataset into a format that can be effectively used to train and test machine learning models.

DATA CLEANING:

Remove duplicates: Eliminate identical or near-identical messages.

Handle missing values: Address any missing data points in the dataset.

FEATURE ENGINEERING:

Convert the preprocessed text data into numerical features that machine learning algorithms can use. Common techniques include:

Bag of Words (BoW): Create a matrix where rows represent documents (messages) and columns represent unique words. The cell values represent the word's frequency in each message.

Term Frequency-Inverse Document Frequency (TF-IDF): Calculate a weight for each word in a document, considering its frequency in the document and rarity across the entire dataset.

Word Embeddings: Utilize pre-trained word vectors (e.g., Word2Vec, GloVe) to represent words in dense vector form.

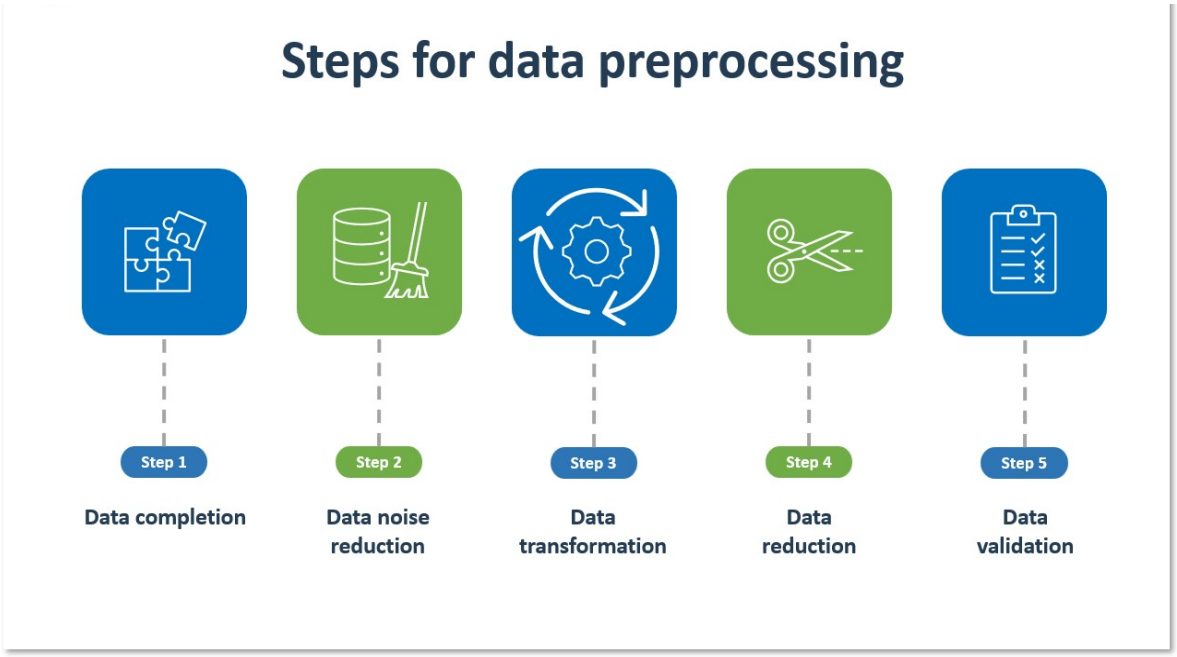
DATA SPLITTING:

Divide the dataset into training and testing subsets to evaluate the model's performance. Common splits include 70/30 or 80/20 for training/testing.

ITERATION AND EXPERIMENTATION:

Experiment with various preprocessing techniques and feature representations to find the most effective approach for your spam classifier.

Each of these preprocessing tasks is essential to ensure that the data used to train the spam classifier is in the best possible format for the machine learning model to learn patterns and make accurate predictions.



PROGRAM:

```
stop_words = stopwords.words('english') + ['u', 'im', 'c']
stemmer = nltk.SnowballStemmer('english')
def clean_text(text):
'''Do lowercase, remove text in square brackets,links,
```

```
punctuation and words containing numbers.''' text =
str(text).lower()
text = re.sub('\[.*?\]', '', text)
text = re.sub('https?://\S+|www\.\S+', '', text)
text = re.sub('<.*?>+', '', text)
text = re.sub('[%s]' % re.escape(string.punctuation), '',
text)
text = re.sub('\n', '', text)
text = re.sub('\w*\d\w*', '', text)
return text
def preprocessing(text):
cleaned_text = clean_text(text)
# remove stopwords
cleaned_text  = ' '.join(word for word in
cleaned_text.split(' ') if word not in stop_words)
# do stem method
cleaned_text  = ' '.join(stemmer.stem(word) for word in
cleaned_text.split(' '))
return cleaned_text
data['Cleaned_Message'] =
data['Message'].apply(preprocessing)
```

```
 # let's show new length after process
 data['New_length'] = data['Cleaned_Message'].apply(lambda
 x: len(x.split(' ')))
 data.head()
 #tokens visualization:
 plt.figure(figsize = (16,5))
 plt.title('Top Words For Ham Message')
 Word_ham = WordCloud(
 background_color=None, mode="RGBA" ,
 width=800,
 height=300,
 )

 Word_ham.generate(' '.join(text for text in  data.loc[
 data['Target']=='ham' , 'Cleaned_Message']))
plt.imshow(Word_ham, interpolation='bilinear')
plt.axis('off')
plt.show()
plt.figure(figsize = (16,5))
plt.title('Top Words For Spam Message')
Word_spam = WordCloud(
```

```
background_color=None, mode="RGBA" ,
width=800,
height=300,
)

Word_spam.generate(' '.join(text for text in  data.loc[
data['Target']=='spam' , 'Cleaned_Message']))
plt.imshow(Word_spam, interpolation='bilinear')
plt.axis('off')
plt.show()
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
Encoder = LabelEncoder()
y_label   = Encoder.fit_transform(data['Target'])
train_data= data['Cleaned_Message']

X_train ,X_test , y_train , y_test =
train_test_split(train_data , y_label , test_size=0.2 ,
random_state=42)
from sklearn.feature_extraction.text import
TfidfTransformer , CountVectorizer
```

```
TF_model = TfidfTransformer()

Vec_model= CountVectorizer()


Vec_model.fit(X_train)

X_train_vec = Vec_model.transform(X_train)

X_test_vec  = Vec_model.transform(X_test)


TF_model.fit(X_train_vec)

X_train_tfidf = TF_model.transform(X_train_vec)

X_test_tfidf  = TF_model.transform(X_test_vec)


X_train_tfidf
```

OUTPUT:

	Target	Message_	Message_length	cleaned_message	New_message

0	ham	Go until jurong point, crazy.. Available only ...	20	go jurong point crazi avail bugi n great world...	16
1	ham	Ok lar... Joking wif u oni...	6	ok lar joke wif oni	5
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	28	free entri wkli comp win fa cup final tkts m...	23
3	ham	U dun say so early hor... U c already then say...	11	dun say earli hor alreadi say	6
4	ham	Nah I don't think he goes to usf, he lives aro...	13	nah dont think goe usf live around though	8

<4457x5955 sparse matrix of type '<class 'numpy.float64'>'

with 34735 stored elements in Compressed Sparse Row format>

CONCLUSION:

In conclusion, loading and preprocessing a spam dataset is a crucial first step in the process of building effective spam email classification models. This phase of data preparation plays a pivotal role in ensuring the accuracy and reliability of the subsequent machine learning or data analysis tasks. Loading and preprocessing the dataset are foundational steps in data analysis and machine learning. These steps ensure that the data is clean, well-structured, and suitable for the specific task at hand. Careful attention to these steps can significantly impact the quality and reliability of the results derived from the data.

The choices made during these steps can significantly impact the quality of your results. It's important to carefully consider the characteristics of the dataset and the specific requirements of your analysis or modeling task to make informed decisions about data loading and preprocessing. Moreover, ongoing iteration and evaluation of these steps are often necessary as you gain insights from the analysis or model performance.