

Abstract:

Nowadays, emails are used in almost every field, from business to education. Emails have two subcategories, i.e., ham and spam. Email spam, also called junk emails or unwanted emails, is a type of email that can be used to harm any user by wasting his/her time, computing resources, and stealing valuable information. The ratio of spam emails is increasing rapidly day by day. Spam detection and filtration are significant and enormous problems for email and IoT service providers nowadays. Among all the techniques developed for detecting and preventing spam, filtering email is one of the most essential and prominent approaches. Several machine learning and deep learning techniques have been used for this purpose, i.e., Naïve Bayes, decision trees, neural networks, and random forest. The project aims to develop a robust system for detecting spam emails and SMS messages using machine learning techniques. We will implement models to classify incoming messages as either spam or legitimate based on their content. The system will leverage Natural Language Processing (NLP) and supervised learning algorithms to achieve high accuracy in distinguishing between spam and non-spam messages.

Introduction:

Spam detection is a crucial task in information security to protect users from unwanted and potentially harmful messages. Email and SMS remain popular communication channels, but they are often targets for spam campaigns. Traditional rule-based filters are limited in their effectiveness, making machine learning-based approaches more appealing due to their ability to adapt and generalize.

Architecture:

The architecture of our spam detection system comprises several key components:

1. Data Preprocessing and Feature Extraction:

- Text Cleaning: Removing HTML tags, special characters, and stopwords.
- Tokenization: Breaking down messages into individual words (tokens).
- Feature Engineering: Using TF-IDF (Term Frequency-Inverse Document Frequency) to convert text into numerical vectors.

2. Machine Learning Models:

- Multinomial Naive Bayes: A probabilistic classifier based on Bayes' theorem with strong performance for text classification tasks.
- Support Vector Machines (SVM): Effective for high-dimensional data, SVM seeks to find the hyperplane that best separates classes.
- Ensemble Methods: Combining multiple models (e.g., Voting Classifier) to improve overall performance.
- Real-time Detection System

3. Integration:

- Deploying the trained models into a real-time application using Flask.
- Continuous Monitoring: Monitoring incoming messages and applying the detection system to classify as spam or not.

Modules:

To implement spam mail detection using machine learning, the modules and libraries required are:

1) For Data Manipulation and Analysis:

Pandas: For data manipulation and analysis, especially for handling tabular data.

NumPy: For numerical operations and array manipulation.

2) For Text Processing:

NLTK (Natural Language Toolkit): For text preprocessing tasks such as tokenization, stemming, and lemmatization.

Scikit-learn's CountVectorizer or TfidfVectorizer: For converting text data into numerical features (Bag of Words or TF-IDF representation).

3) For Machine Learning Models:

Scikit-learn: A comprehensive library for machine learning in Python. It provides various classifiers such as Naive Bayes, SVM, Logistic Regression, Random Forest, and GBM.

TensorFlow or PyTorch: For building and training deep learning models, though they might be overkill for simpler spam detection tasks.

4) For Model Evaluation:

Scikit-learn: Provides functions for evaluating the performance of machine learning models, including metrics like accuracy, precision, recall, F1-score, and ROC-AUC curve.

Implementation:

Import numpy, panda, matplotlib.pyplot and seaborn libraries. Read a CSV file named "spam.csv" using the given below command. Display the first 5 rows of the DataFrame 'df'.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd

# Try reading the file with different encodings
try:
    df = pd.read_csv("spam.csv", encoding='utf-8')
except UnicodeDecodeError:
    try:
        df = pd.read_csv("spam.csv", encoding='latin1')
    except UnicodeDecodeError:
        df = pd.read_csv("spam.csv", encoding='ISO-8859-1')
```

1. Data cleaning:

Display the index, columns, data types, and memory usage of the dataframe 'df'.

```
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
```

Drop the last three columns.

```
3]: # drop last 3 cols
    df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True)
```

Get a random sample of data from the DataFrame

```
df.sample(5)
```

```
      v1                                     v2
3876  ham  Sorry sir, i will call you tomorrow.  senthil...
4548  ham  Haha, my friend tyler literally just asked if ...
2799  ham  House-Maid is the murderer, coz the man was mu...
1352  ham  Let Ur Heart Be Ur Compass Ur Mind Ur Map Ur S...
3443  ham           Yes but I don't care cause I know its there!
```

Import LabelEncoder from sklearn.preprocessing. Assign numerical labels to the different categories present in the 'target' column using fit_transform method.

```
from sklearn.preprocessing import LabelEncoder  
encoder = LabelEncoder()
```

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

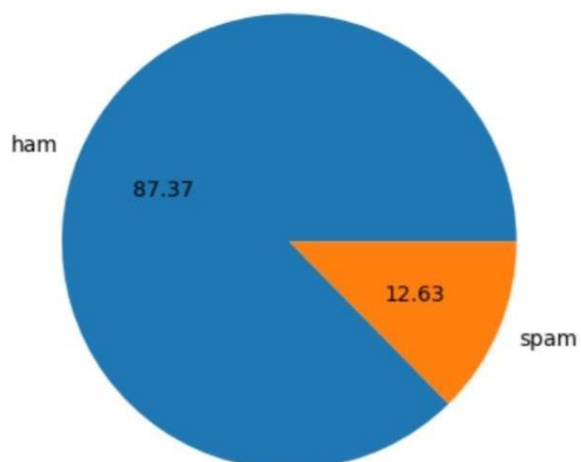
```
df.head()
```

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

2. EDA

By using matplotlib, create a pie chart based on the counts of unique values in the 'target' column of the DataFrame **df**.

```
import matplotlib.pyplot as plt  
plt.pie(df['target'].value_counts(), labels=['ham', 'spam'], autopct="%0.2f")  
plt.show()
```



Calculate the number of words, characters and sentences and apply the given functions.

```
: # num of words
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
```

```
: df.head()
```

```
:      target      text  num_characters
0         0  Go until jurong point, crazy.. Available only ...      111
1         0              Ok lar... Joking wif u oni...      29
2         1  Free entry in 2 a wkly comp to win FA Cup fina...     155
3         0  U dun say so early hor... U c already then say...     49
4         0  Nah I don't think he goes to usf, he lives aro...     61
```

```
      num_words
0           24
1            8
2           37
3           13
4           15
```

```
df['num_sentences'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

```
df.head()
```

```
      target      text  num_characters
0         0  Go until jurong point, crazy.. Available only ...      111
1         0              Ok lar... Joking wif u oni...      29
2         1  Free entry in 2 a wkly comp to win FA Cup fina...     155
3         0  U dun say so early hor... U c already then say...     49
4         0  Nah I don't think he goes to usf, he lives aro...     61
```

```
      num_words  num_sentences
0           24             2
1            8             2
2           37             2
3           13             1
4           15             1
```

```
df[['num_characters', 'num_words', 'num_sentences']].describe()
```

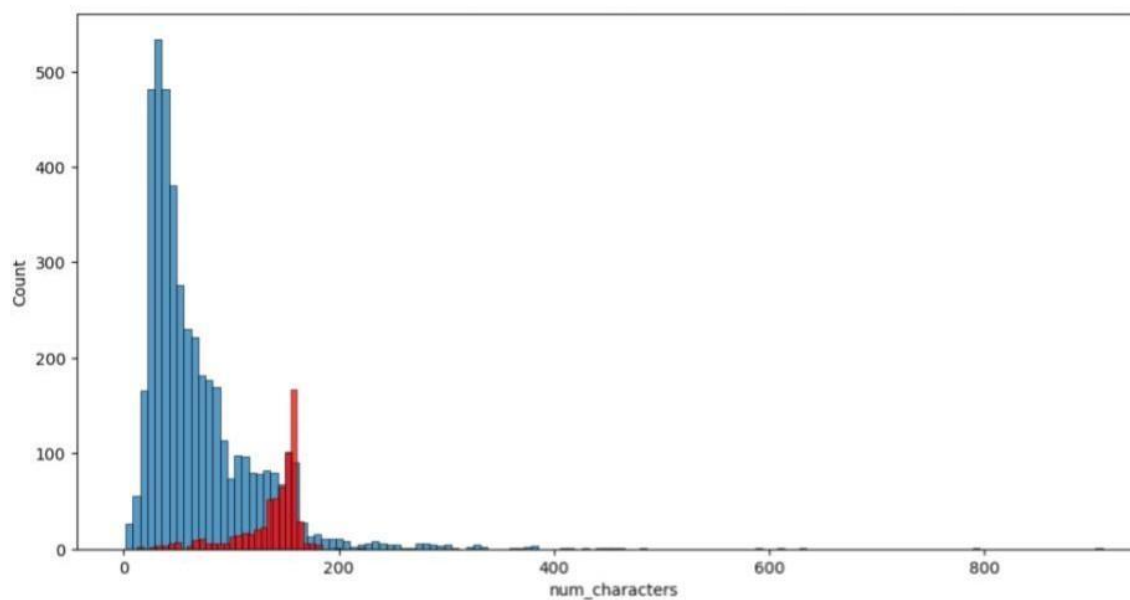
```
      num_characters  num_words  num_sentences
count      5169.000000    5169.000000    5169.000000
mean         78.977945     18.455794      1.965564
std         58.236293     13.324758      1.448541
min          2.000000      1.000000      1.000000
25%         36.000000      9.000000      1.000000
50%         60.000000     15.000000      1.000000
75%        117.000000     26.000000      2.000000
max        910.000000    220.000000     38.000000
```

Create a histogram of the distribution of the number of characters in the 'text' column, separated by the 'target' column values (0 and 1, likely representing 'ham' and 'spam' categories) by using seaborn and matplotlib

```
import seaborn as sns
```

```
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_characters'])
sns.histplot(df[df['target'] == 1]['num_characters'],color='red')
```

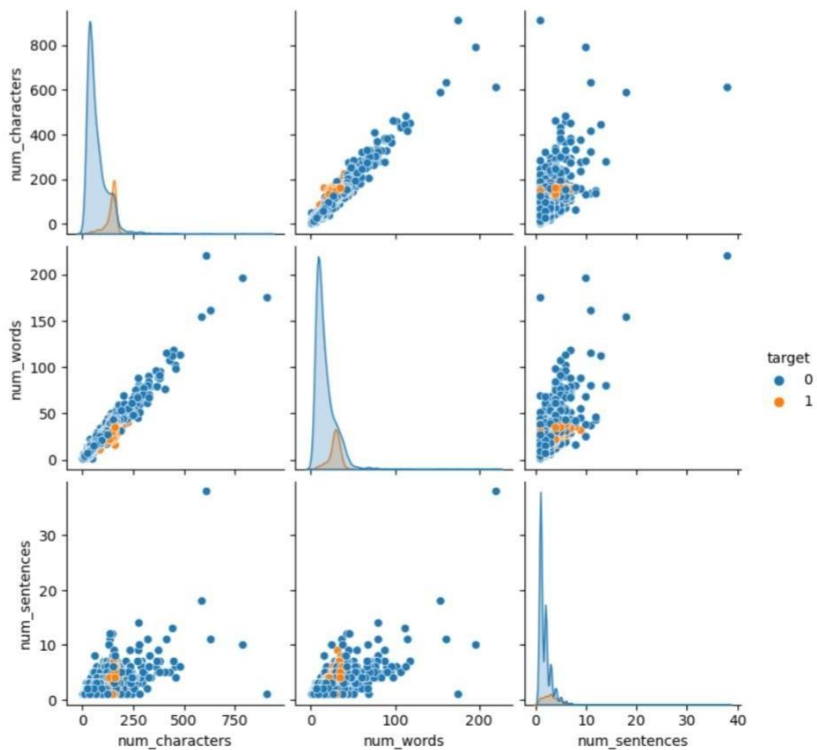
```
<Axes: xlabel='num_characters', ylabel='Count'>
```



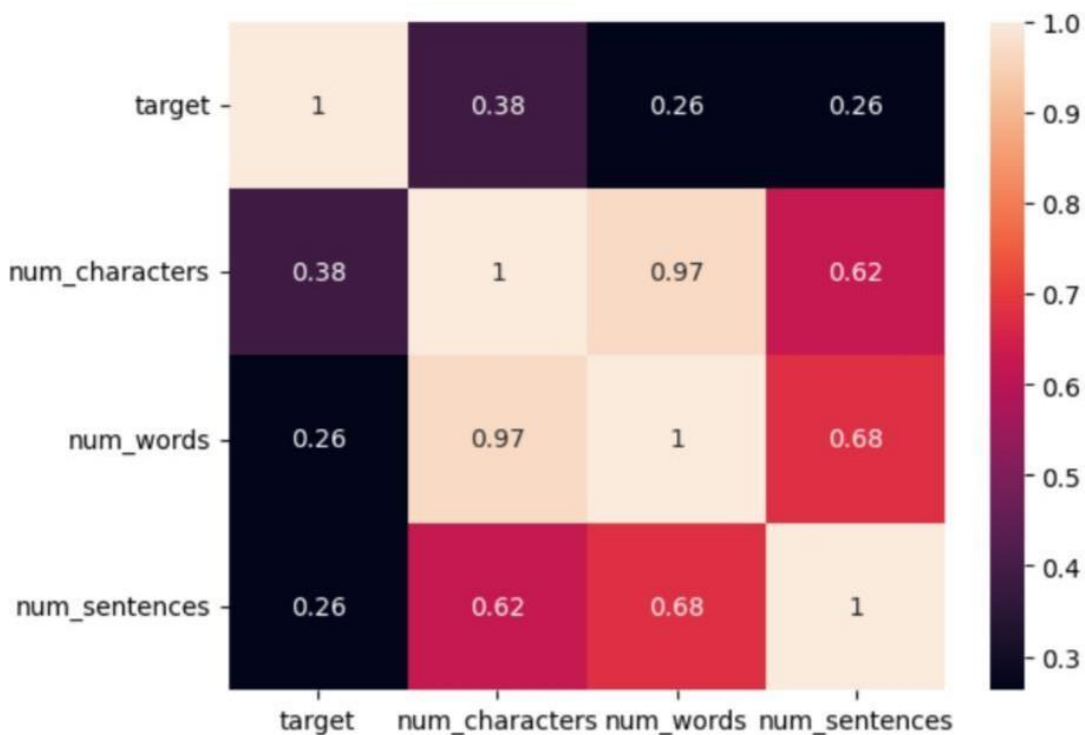
```
[47]: plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_words'])
sns.histplot(df[df['target'] == 1]['num_words'],color='red')
```

```
[47]: <Axes: xlabel='num_words', ylabel='Count'>
```

Create a pairplot using seaborn with the 'target' variable as the hue



Generate a heatmap using seaborn to visualize the correlation matrix of your DataFrame, with annotations displaying the correlation values `sns.heatmap(df.corr(),annot=True)`



3. Data reprocessing

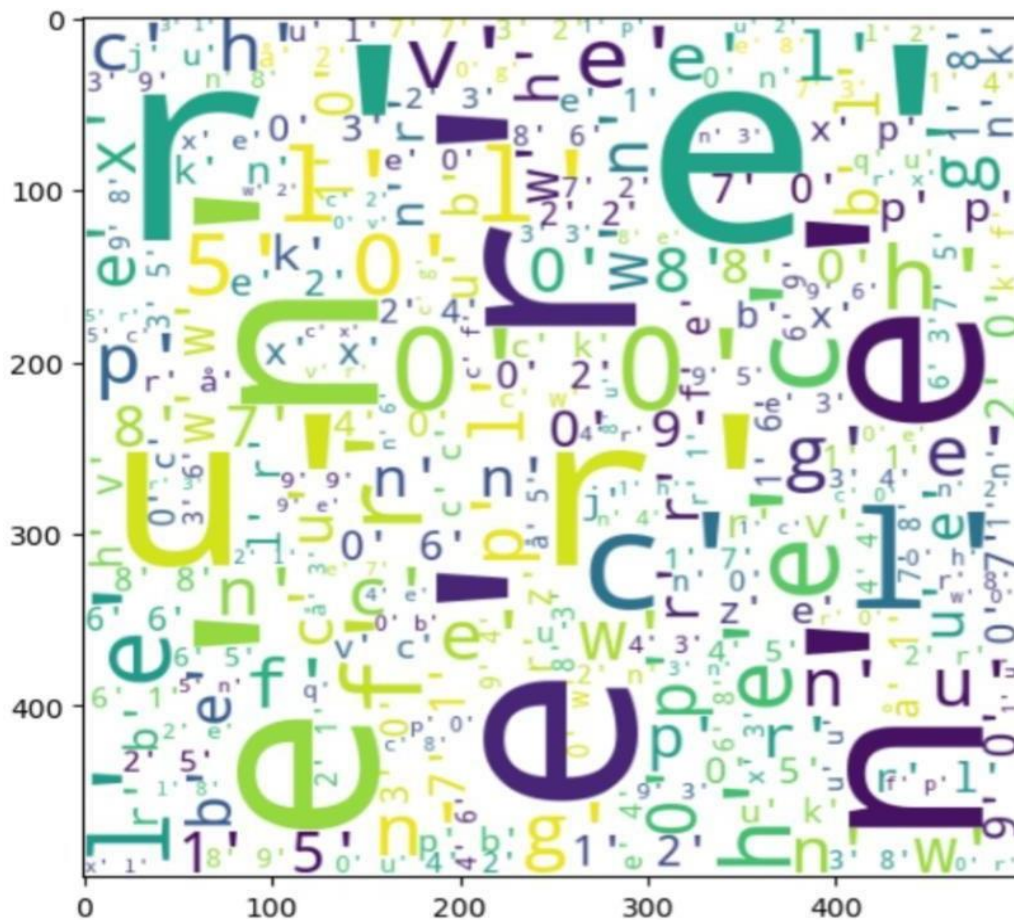
Define a function `transform_text` to preprocess text data.

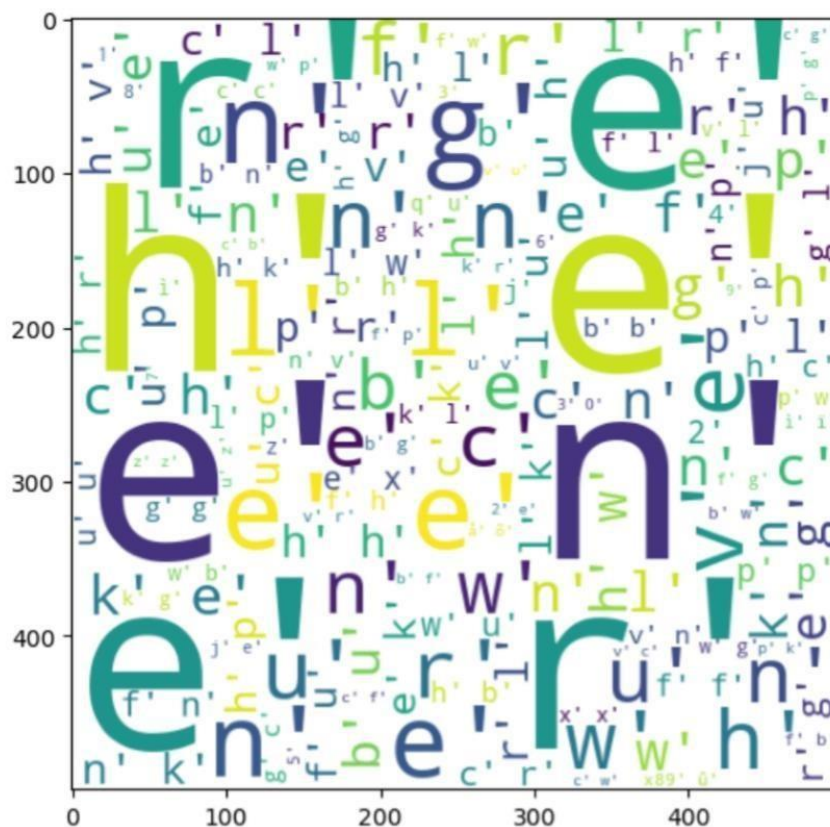
```
import string
from nltk.corpus import stopwords

def transform_text(text):
    y = []
    # Ensure all characters are lowercase for uniformity
    text = text.lower()
    y.clear()
    for i in text:
        # Check if the character is not a stopword or punctuation
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    text = y[:]
    return text

transform_text("I'm gonna be home soon and i don't want to talk about this_
stuff anymore tonight, k? I've cried enough today.")
```

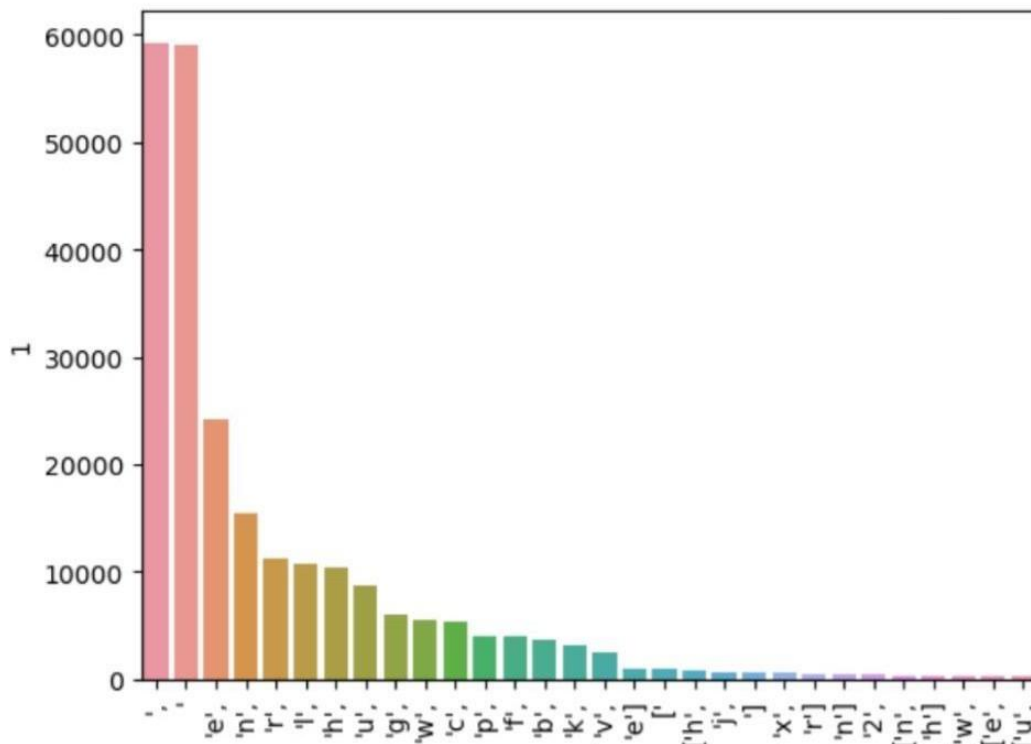
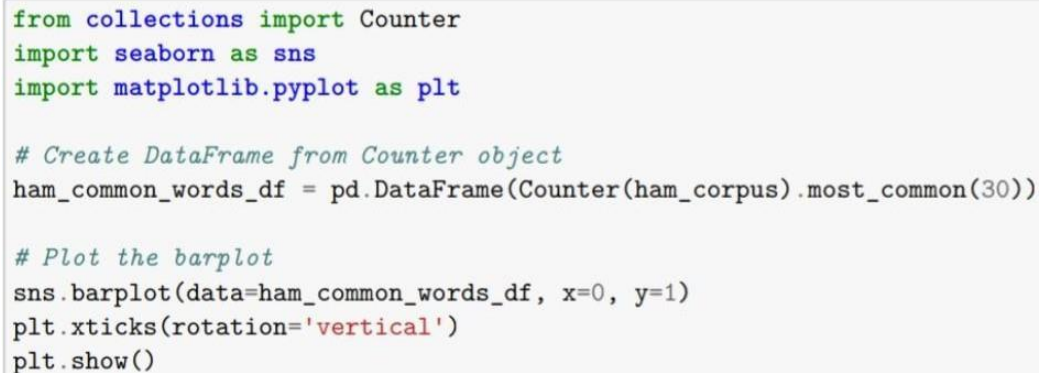
Generate a word cloud from the 'transformed_text' column in the DataFrame, specifically for the 'target' category equal to 1.





```
len(spam_corpus)
```

Build a corpus of words by iterating through messages in the DataFrame where target = spam. Generate a bar plot for the most common words found in non-spam (ham) messages.



Involves text preprocessing and vectorization using Bag of Words (BoW) methodology on a DataFrame containing text data for machine learning tasks like spam detection.

```
[85]: # Text Vectorization
# using Bag of Words
df.head()
```

```
[85]:   target      text  num_characters \
0      0  Go until jurong point, crazy.. Available only ...      111
1      0              Ok lar... Joking wif u oni...       29
2      1  Free entry in 2 a wkly comp to win FA Cup fina...     155
3      0  U dun say so early hor... U c already then say...     49
4      0  Nah I don't think he goes to usf, he lives aro...     61

   num_words  num_sentences  transformed_text
0         24             2  ['g', ' ', 'u', 'n', 'l', ' ', 'j', 'u', 'r', ...
1          8             2  ['k', ' ', 'l', 'r', ' ', 'j', 'k', 'n', 'g', ...
2         37             2  ['f', 'r', 'e', 'e', ' ', 'e', 'n', 'r', ' ', ...
3         13             1  ['u', ' ', 'u', 'n', ' ', ' ', ' ', 'e', 'r', ...
4         15             1  ['n', 'h', ' ', ' ', 'n', ' ', 'h', 'n', 'k', ...
```

4. Model Building

```
[86]: from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

```
[87]: X = tfidf.fit_transform(df['transformed_text']).toarray()
```

```
[88]: #from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler()
#X = scaler.fit_transform(X)
```

```
[89]: # appending the num_character col to X
#X = np.hstack((X,df['num_characters'].values.reshape(-1,1)))
```

```
[90]: X.shape
```

```
[90]: (5169, 3)
```

The code performs classification using Gaussian Naive Bayes (GaussianNB) on the training data (**X_train**, **y_train**) and evaluates the model's accuracy, confusion matrix, and precision score on the test data (**X_test**, **y_test**).

```

[91]: y = df['target'].values

[92]: from sklearn.model_selection import train_test_split

[93]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
      ↪2,random_state=2)

[94]: from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
      from sklearn.metrics import accuracy_score,confusion_matrix,precision_score


[95]: gnb = GaussianNB()
      mnb = MultinomialNB()
      bnb = BernoulliNB()

[96]: gnb.fit(X_train,y_train)
      y_pred1 = gnb.predict(X_test)
      print(accuracy_score(y_test,y_pred1))
      print(confusion_matrix(y_test,y_pred1))
      print(precision_score(y_test,y_pred1))

0.14313346228239845
[[ 10 886]
 [  0 138]]
0.134765625

```

This code calculates the accuracy, confusion matrix, and precision score for a classification model (**y_pred**) compared to the true labels (**y_test**). The precision score is explicitly set to 0 for classes with no predicted samples (using **zero_division=0**) to avoid warnings.

- **Accuracy:** 86.65%
- **Confusion Matrix:**

[89601380][89613800]

- 896 true negatives (predicted as 0, actual 0)
- 0 false positives (predicted as 1, actual 0)
- 138 false negatives (predicted as 0, actual 1)
- 0 true positives (predicted as 1, actual 1)
- **Precision:** 0.0

```
[98]: from sklearn.naive_bayes import MultinomialNB
      from sklearn.metrics import accuracy_score, confusion_matrix, precision_score

      # Initialize Multinomial Naive Bayes classifier
      mnb = MultinomialNB()

      # Fit the classifier on training data
      mnb.fit(X_train, y_train)

      # Predict labels on test data
      y_pred = mnb.predict(X_test)

      # Calculate evaluation metrics
      accuracy = accuracy_score(y_test, y_pred)
      conf_matrix = confusion_matrix(y_test, y_pred)
      precision = precision_score(y_test, y_pred, zero_division=0) # Set
      ↪ zero_division to 0 to avoid warnings

      # Print evaluation metrics
      print("Accuracy:", accuracy)
      print("Confusion Matrix:\n", conf_matrix)
      print("Precision:", precision)
```

```
Accuracy: 0.8665377176015474
Confusion Matrix:
[[896  0]
 [138  0]]
Precision: 0.0
```

```
[100]: accuracy = accuracy_score(y_test, y_pred)
       conf_matrix = confusion_matrix(y_test, y_pred)
       precision = precision_score(y_test, y_pred, zero_division=0) # Set
       ↪ zero_division to 0 to avoid warnings
```

```
Accuracy: 0.8665377176015474
Confusion Matrix:
[[896  0]
 [138  0]]
Precision: 0.0
```

This code snippet defines multiple classifiers from different machine learning algorithms, and stores them in a dictionary (**clfs**) for model evaluation purposes.


```

[101]: # tfidf --> MNB

[108]: from sklearn.linear_model import LogisticRegression
      from sklearn.svm import SVC
      from sklearn.naive_bayes import MultinomialNB
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.ensemble import AdaBoostClassifier
      from sklearn.ensemble import BaggingClassifier
      from sklearn.ensemble import ExtraTreesClassifier
      from sklearn.ensemble import GradientBoostingClassifier
      from xgboost import XGBClassifier

[109]: svc = SVC(kernel='sigmoid', gamma=1.0)
      knc = KNeighborsClassifier()
      mnb = MultinomialNB()
      dtc = DecisionTreeClassifier(max_depth=5)
      lrc = LogisticRegression(solver='liblinear', penalty='l1')
      rfc = RandomForestClassifier(n_estimators=50, random_state=2)
      abc = AdaBoostClassifier(n_estimators=50, random_state=2)
      bc = BaggingClassifier(n_estimators=50, random_state=2)
      etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
      gbdt = GradientBoostingClassifier(n_estimators=50, random_state=2)
      xgb = XGBClassifier(n_estimators=50, random_state=2)

[110]: clfs = {
      'SVC' : svc,
      'KN' : knc,
      'NB': mnb,
      'DT': dtc,
      'LR': lrc,
      'RF': rfc,
      'AdaBoost': abc,
      'BgC': bc,
      'ETC': etc,

```

This code creates a tidy dataframe (**performance_df1**) by melting the original dataframe (**performance_df**) to have a variable column (**variable**) that includes 'Accuracy' and 'Precision' values corresponding to each algorithm (**Algorithm**)


```
: performance_df
```

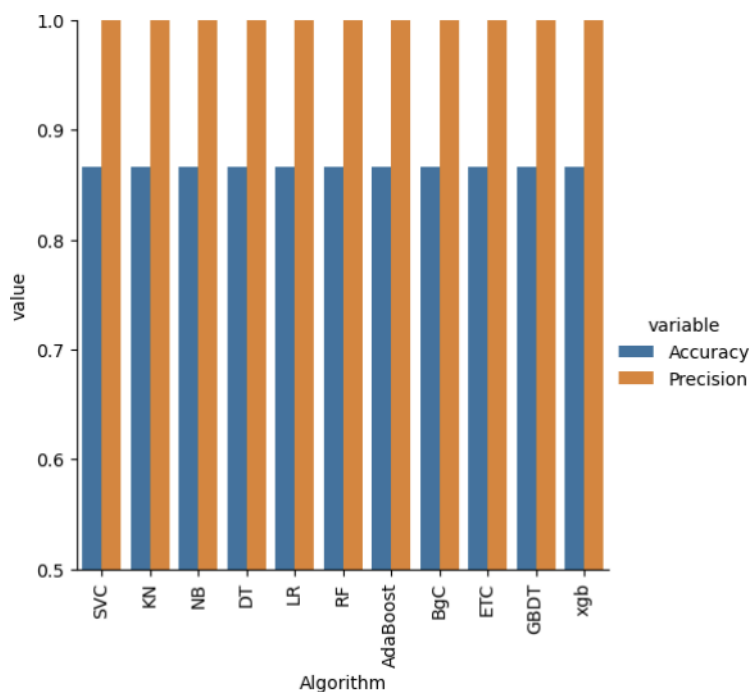
```
:   Algorithm  Accuracy  Precision
0         SVC  0.866538         1.0
1         KN  0.866538         1.0
2         NB  0.866538         1.0
3         DT  0.866538         1.0
4         LR  0.866538         1.0
5         RF  0.866538         1.0
6   AdaBoost  0.866538         1.0
7         BgC  0.866538         1.0
8         ETC  0.866538         1.0
9         GBDT 0.866538         1.0
10        xgb  0.866538         1.0
```

```
: performance_df1 = pd.melt(performance_df, id_vars = "Algorithm")
```

```
: performance_df1
```

```
:   Algorithm  variable    value
0         SVC  Accuracy  0.866538
1         KN  Accuracy  0.866538
```

```
sns.catplot(x = 'Algorithm', y='value',
            hue = 'variable', data=performance_df1, kind='bar', height=5)
plt.ylim(0.5,1.0)
plt.xticks(rotation='vertical')
plt.show()
```



The code demonstrates merging two dataframes (**new_df_scaled** and **temp_df**) on the 'Algorithm' column. It then constructs a Voting Classifier using Support Vector Machine (SVC), Multinomial Naive Bayes (MNB), and Extra Trees Classifier (ETC) as estimators with soft voting. Additionally, it applies a Stacking Classifier using the same estimators and a Random Forest Classifier as the final estimator.

```
[135]: new_df_scaled.merge(temp_df,on='Algorithm')
```

```
[135]:
```

	Algorithm	Accuracy	Precision	Accuracy_scaling_x	Precision_scaling_x \
0	SVC	0.866538	1.0	0.866538	1.0
1	KN	0.866538	1.0	0.866538	1.0
2	NB	0.866538	1.0	0.866538	1.0
3	DT	0.866538	1.0	0.866538	1.0
4	LR	0.866538	1.0	0.866538	1.0
5	RF	0.866538	1.0	0.866538	1.0
6	AdaBoost	0.866538	1.0	0.866538	1.0
7	BgC	0.866538	1.0	0.866538	1.0
8	ETC	0.866538	1.0	0.866538	1.0
9	GBDT	0.866538	1.0	0.866538	1.0
10	xgb	0.866538	1.0	0.866538	1.0

	Accuracy_scaling_y	Precision_scaling_y	Accuracy_num_chars \
0	0.866538	1.0	0.866538
1	0.866538	1.0	0.866538
2	0.866538	1.0	0.866538
3	0.866538	1.0	0.866538
4	0.866538	1.0	0.866538
5	0.866538	1.0	0.866538
6	0.866538	1.0	0.866538
7	0.866538	1.0	0.866538
8	0.866538	1.0	0.866538
9	0.866538	1.0	0.866538
10	0.866538	1.0	0.866538

	Precision_num_chars
0	1.0
1	1.0
2	1.0
3	1.0
4	1.0
5	1.0
6	1.0
7	1.0
8	1.0
9	1.0
10	1.0

Both classifiers are trained on the provided training data (**X_train** and **y_train**) and evaluated using accuracy and precision metrics on the test data (**X_test** and **y_test**). The warning about undefined precision is due to zero predicted samples. Finally, the trained TF-IDF vectorizer (**tfidf**) and Multinomial Naive Bayes model (**mnb**) are serialized using pickle for future use.

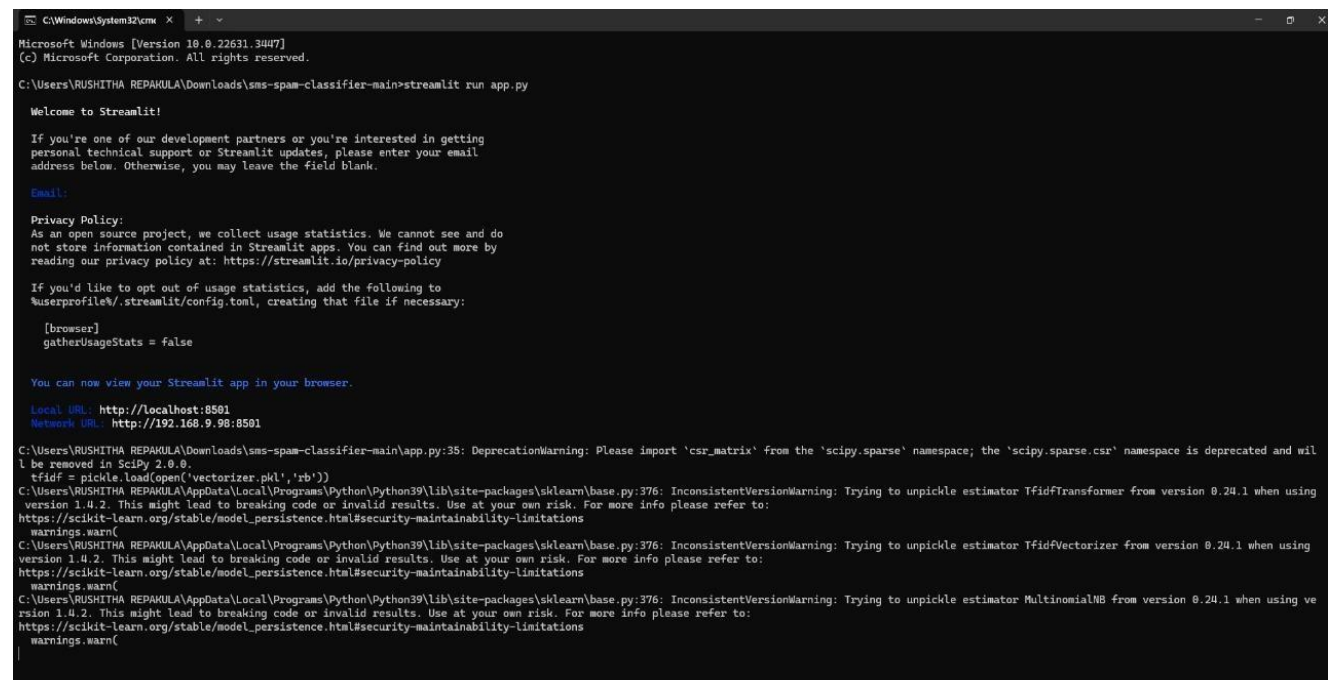
```
[142]: clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
```

```
[143]: clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

Accuracy 0.8665377176015474

Precision 0.0

Testing:



```
C:\Windows\System32\cmd
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\RUSHITHA REPAKULA\Downloads\sms-spam-classifier-main>streamlit run app.py

Welcome to Streamlit!

If you're one of our development partners or you're interested in getting
personal technical support or Streamlit updates, please enter your email
address below. Otherwise, you may leave the field blank.

Email:

Privacy Policy:
As an open source project, we collect usage statistics. We cannot see and do
not store information contained in Streamlit apps. You can find out more by
reading our privacy policy at: https://streamlit.io/privacy-policy

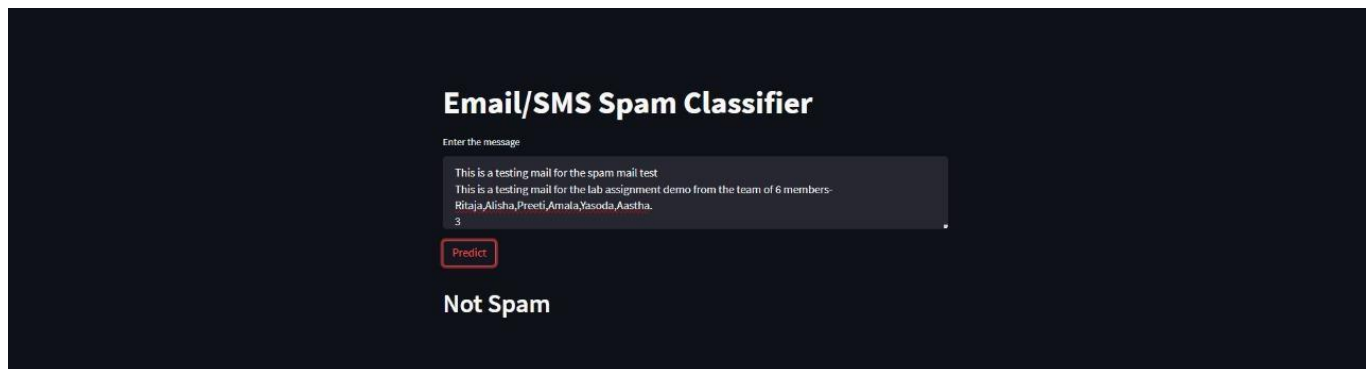
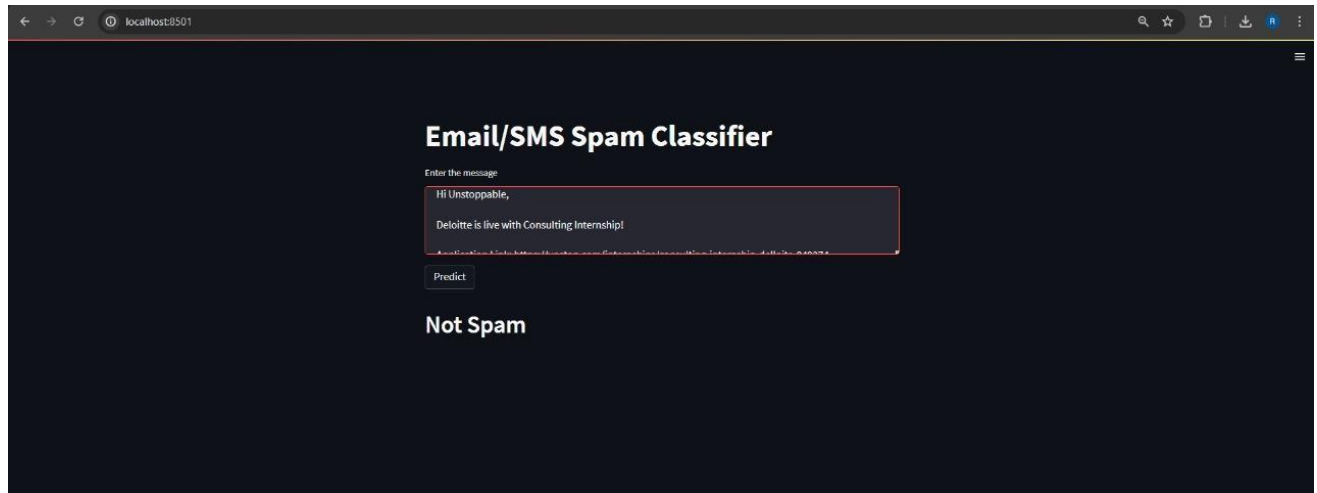
If you'd like to opt out of usage statistics, add the following to
%userprofile%\.streamlit/config.toml, creating that file if necessary:

[browser]
gatherUsageStats = false

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.9.98:8501

C:\Users\RUSHITHA REPAKULA\Downloads\sms-spam-classifier-main\app.py:35: DeprecationWarning: Please import 'csr_matrix' from the 'scipy.sparse' namespace; the 'scipy.sparse.csr' namespace is deprecated and will be removed in SciPy 2.0.0.
  tfidf = pickle.load(open('vectorizer.pkl', 'rb'))
C:\Users\RUSHITHA REPAKULA\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator TfidfTransformer from version 0.24.1 when using version 1.4.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\RUSHITHA REPAKULA\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator TfidfVectorizer from version 0.24.1 when using version 1.4.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\RUSHITHA REPAKULA\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator MultinomialNB from version 0.24.1 when using version 1.4.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
```

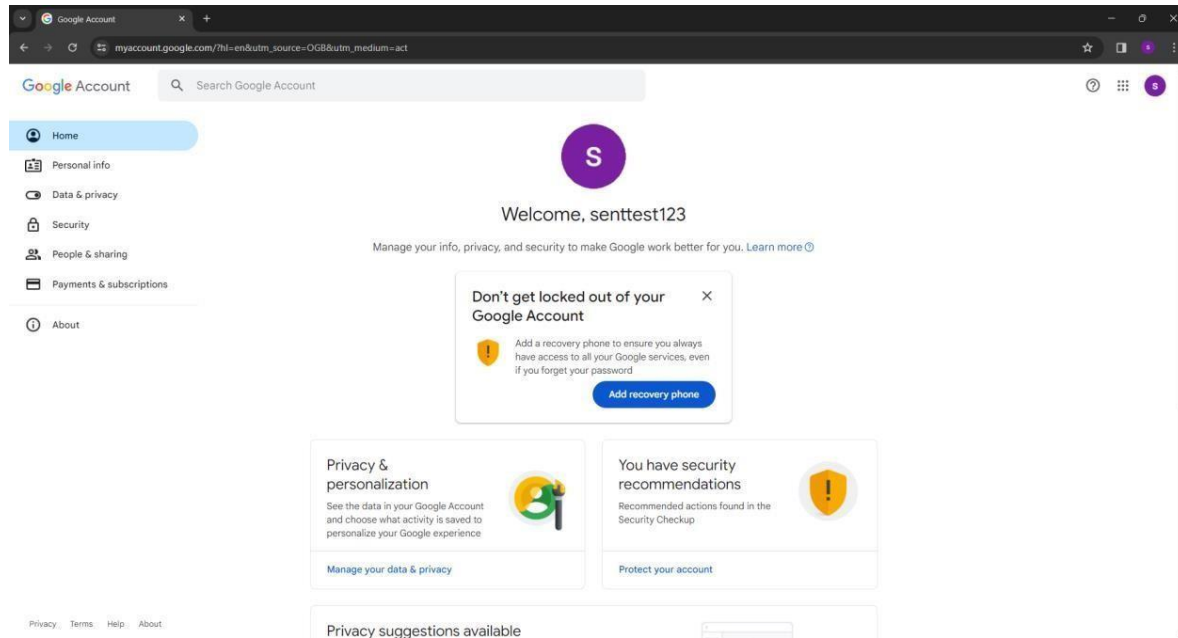


Sending Spam Email

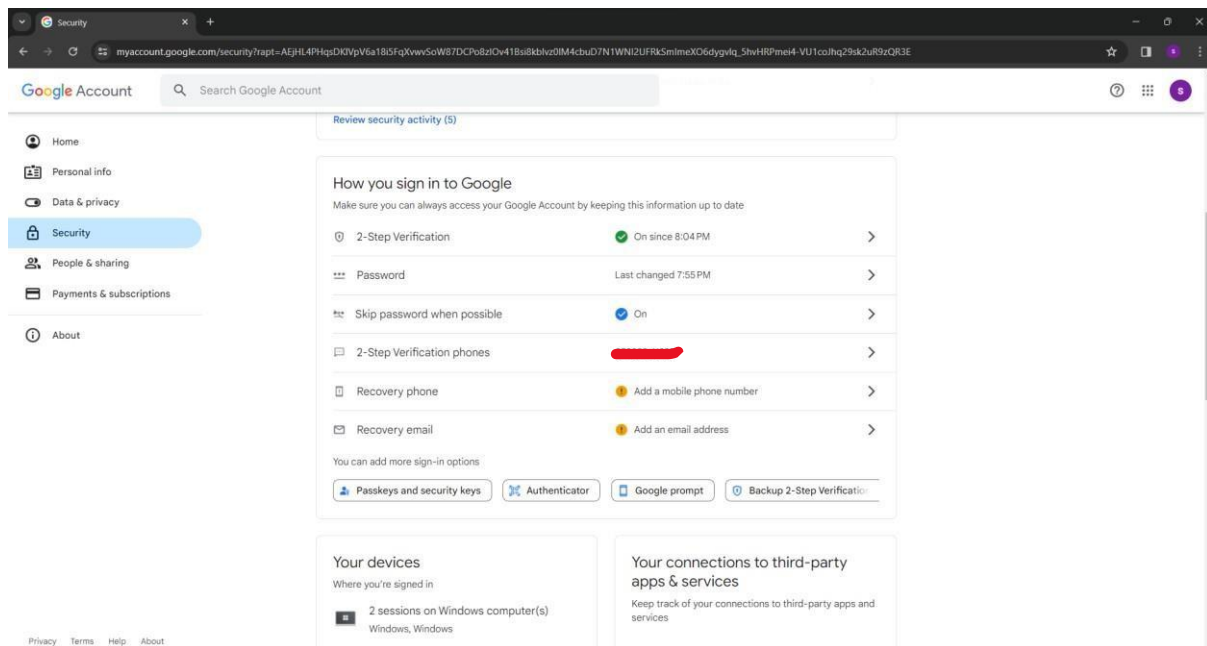
- Sending spam email from duplicate Gmail to a duplicate Gmail that have been created.

Sender Gmail: senttest43@gmail.com

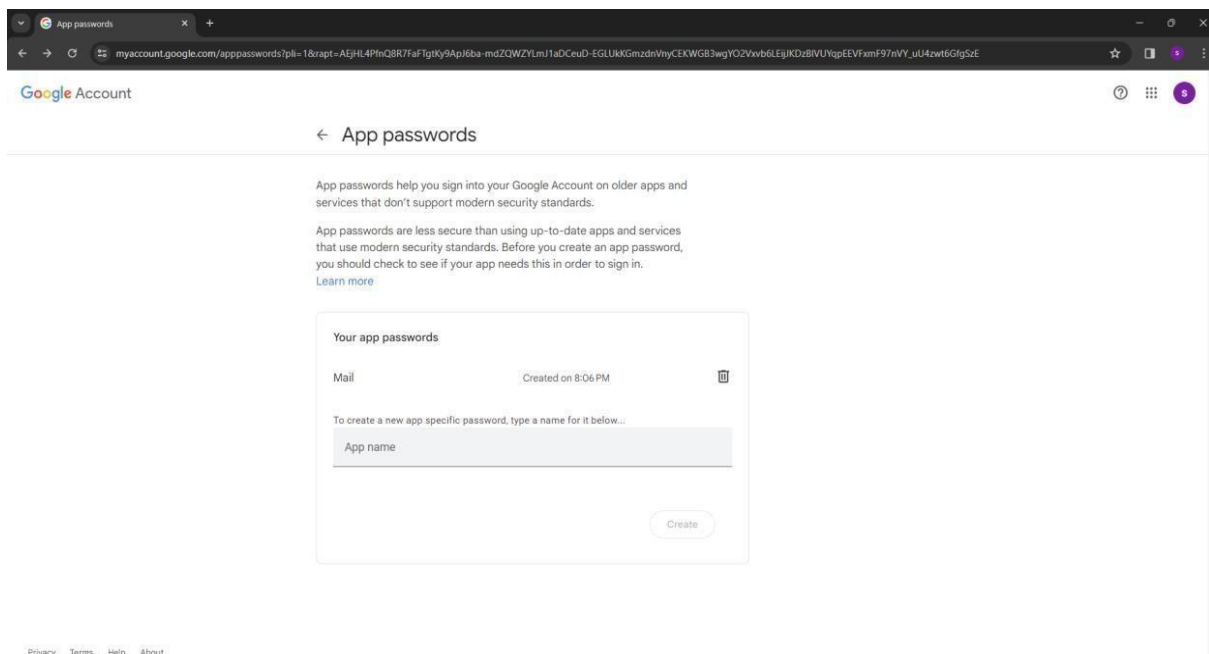
Receiver Gmail: receivetest123@gmail.com



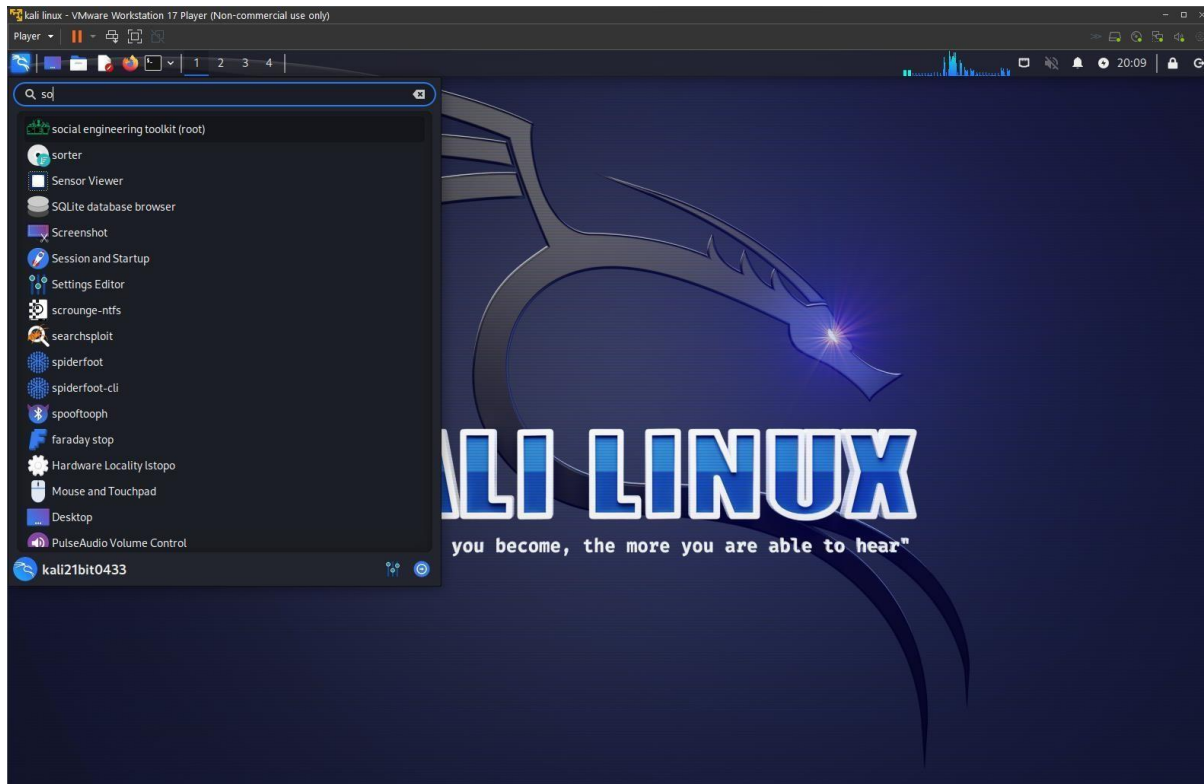
➤ Changing the setting in the sender Gmail for switching on the two-step verification



➤ Creating an App password for sending the spam email from the kali Linux. This password is used in the kali Linux while sending the email to the receiver.



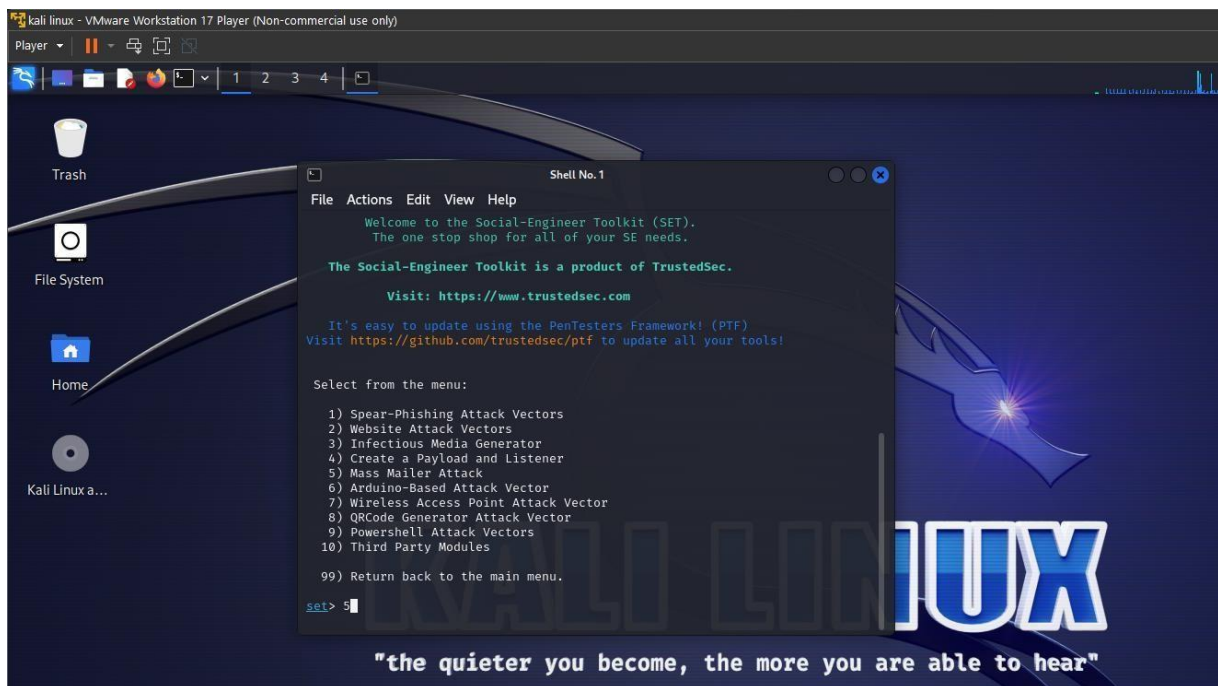
- Open kali Linux and search for social engineering toolkit in the applications.



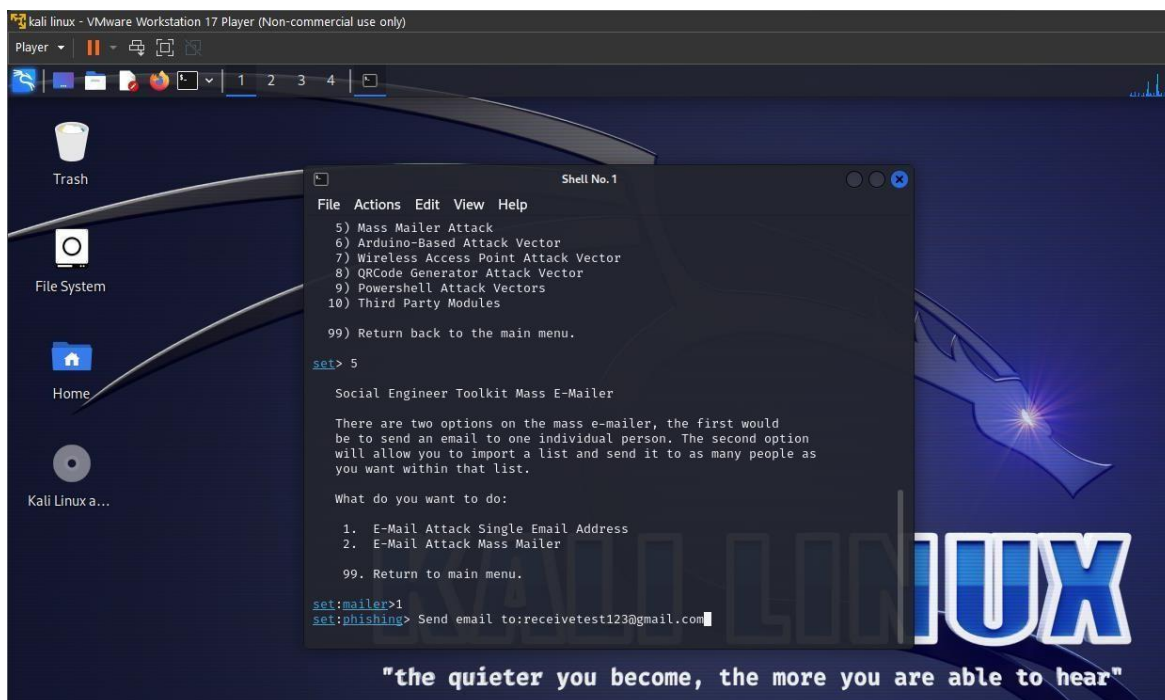
- Open the social engineering tool kit and select option 1 i.e. Social-Engineering attacks



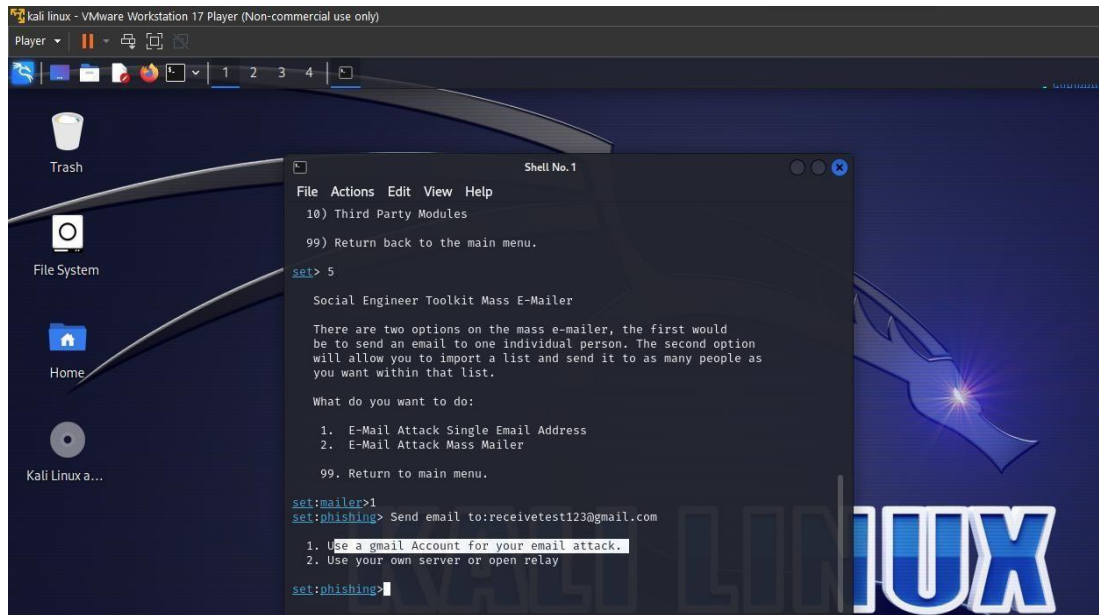
- Now select option 5 i.e. Mass Mailer Attack



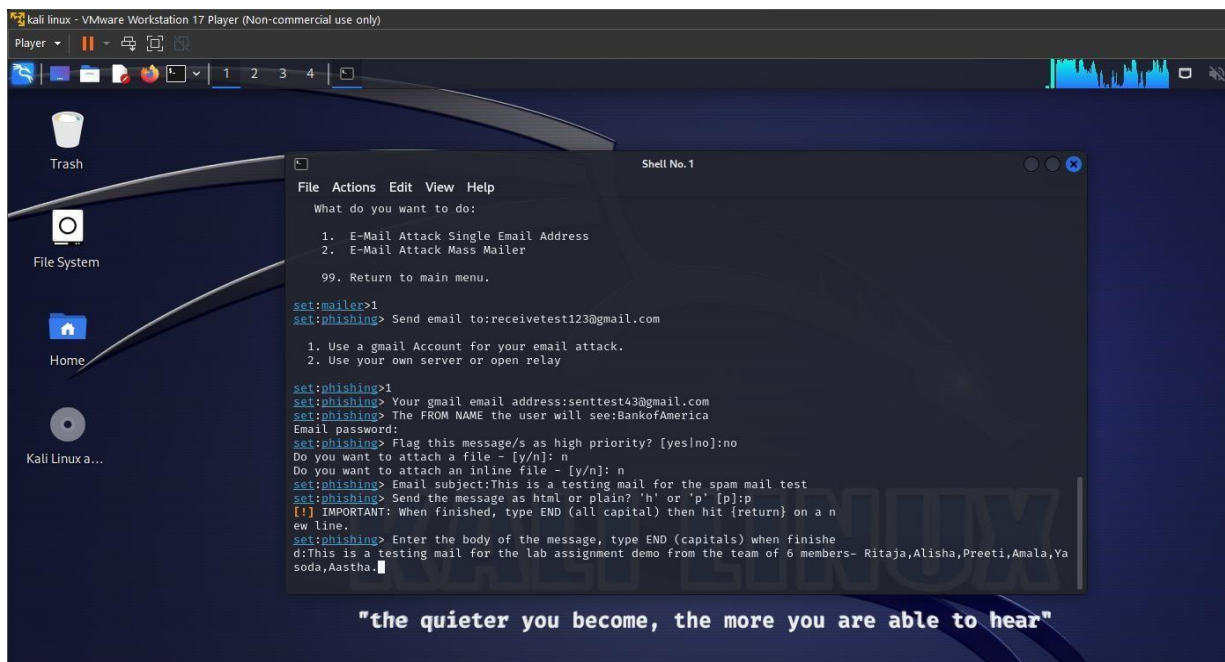
- Now select option 1 i.e. E-mail attack single email address.
- Now it asks for the receiver's email. Give the mail id that has to be received email.



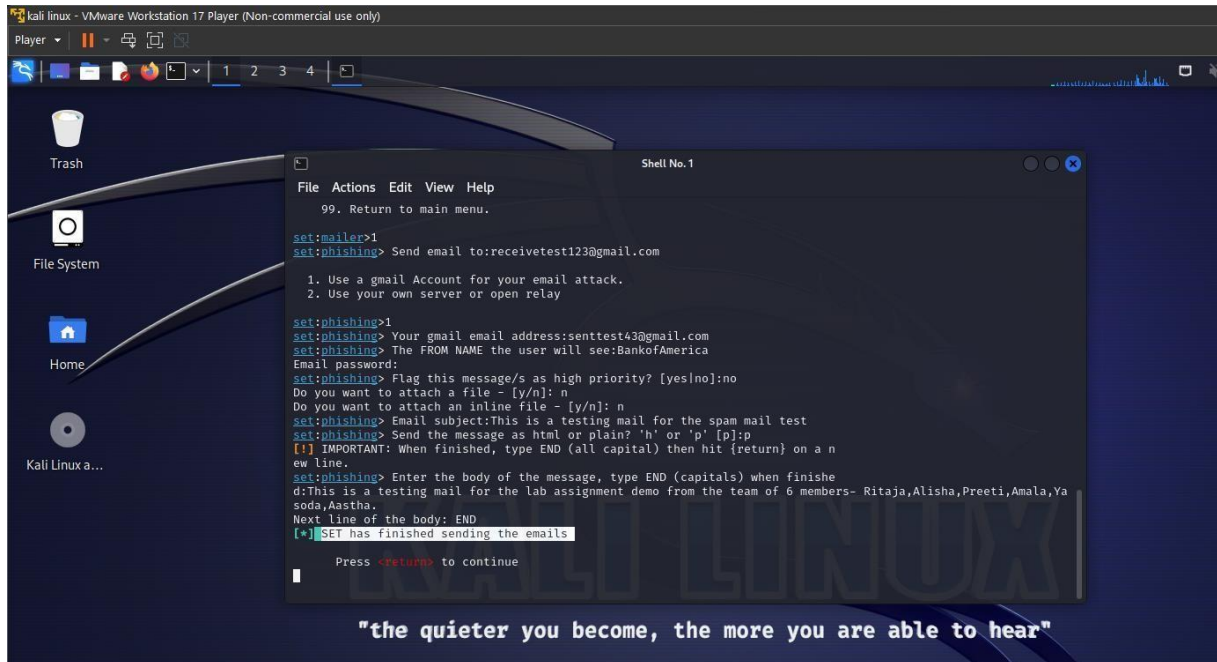
- Now it asks for the mode of email sending. Two options of sending one is using an email address and the other is using a server. Here we selection the option 1 i.e. use a Gmail account for your email attack.



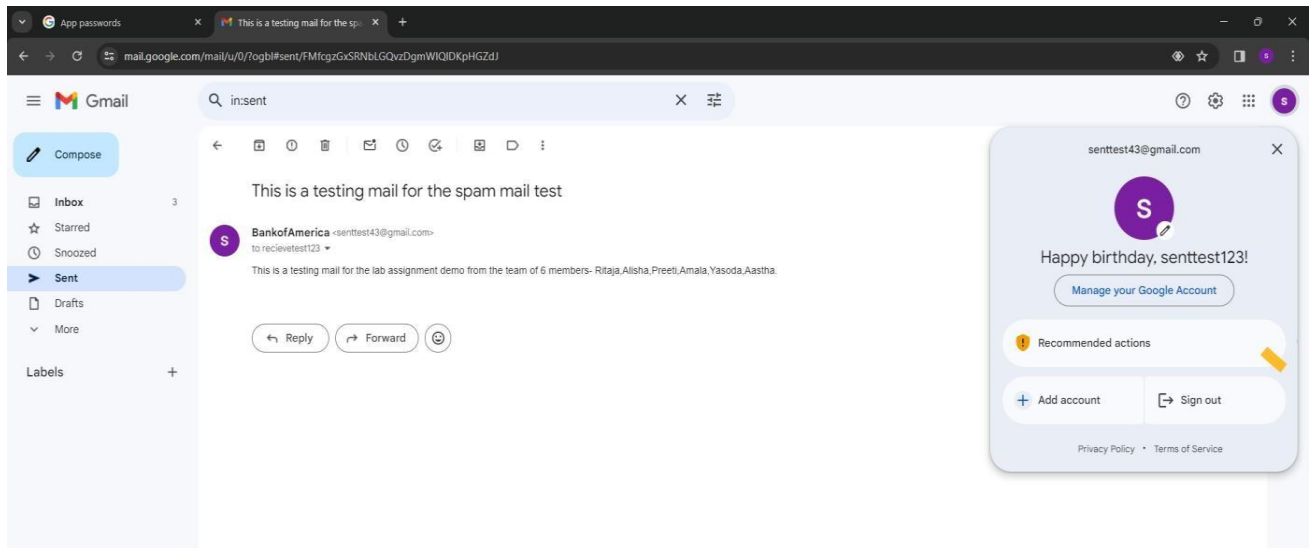
- It asks for the Gmail of the sender.
- It asks for the information that has to be sent in the email.
- They are FROM NAME the user will see, file attachments, Email subject, body of the message.
- It also asks for the option whether the message should be sent in the html format or as a plain.



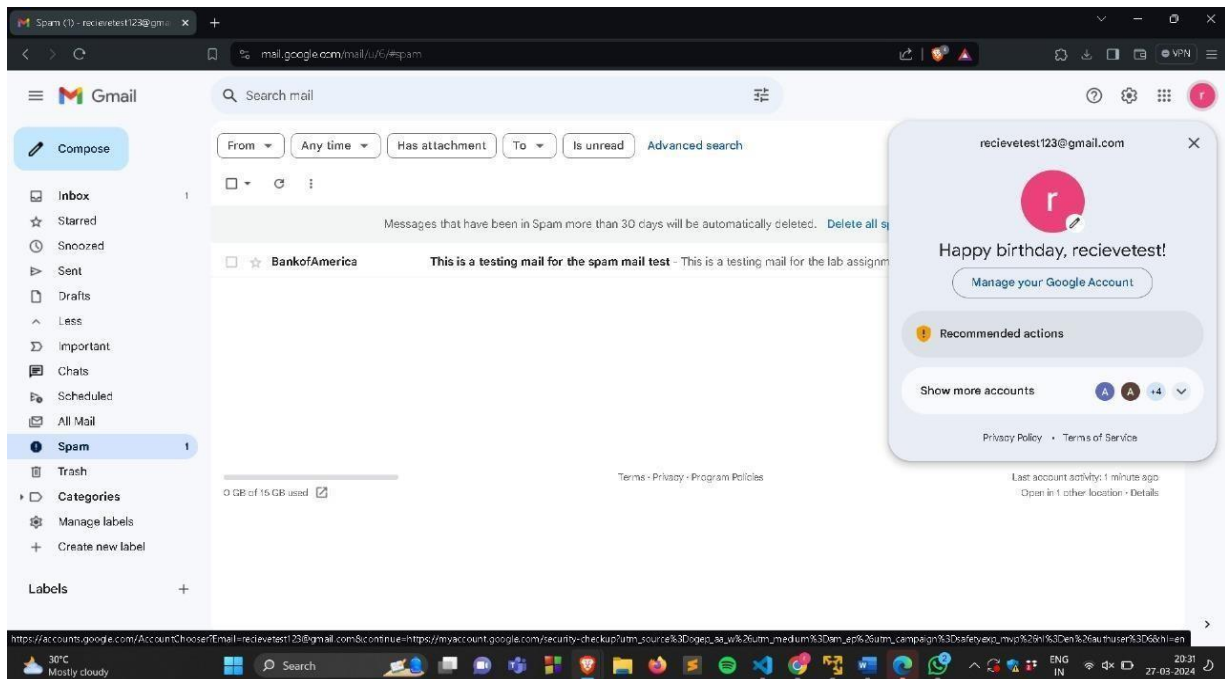
- Type END when you are finished with the body of the message.



- The mail is sent to the receiver email and can be seen in the sent box of the sender email.



- The mail is received to the receivers Gmail and it can be seen in the spam section.



Conclusion:

The implementation of a spam detection system using machine learning models such as Naive Bayes, SVM, and ensemble methods demonstrates the effectiveness of these techniques in classifying spam and non-spam messages. By integrating the models into a real-time detection system using Flask, we create a practical application that can be deployed to protect users from unwanted messages.

This expanded implementation plan includes detailed code snippets for data preprocessing, model training, evaluation, and integration into a Flask-based real-time detection system. Adjust and customize the code to fit your specific project requirements and dataset. The provided architecture and modules outline a structured approach towards building an effective spam detection system.

References:

- 1) Priti Sharma, Uma Bhardwaj, "Machine Learning based Spam E-Mail Detection", Nov 14, 2017
- 2) Awad, W. A., & Elseuofi, S. M. (2011). Machine Learning Methods for Spam E-Mail Classification. AIRCC's International Journal of Computer Science and Information Technology, 3(1), 173–184.