
BITS Pilani
Dubai Campus

# SMS SPAM CLASSIFIER

**GROUP MEMBERS:**

**FATHIMATH RIFNA: 2020A7PS0177U**

**KENZIE SHARON MELCHI JOSEPH: 2020A7PS0166U**

**POULIN MARIA SHAJU: 2020A7PS0072U**

**COURSE: ARTIFICIAL INTELLIGENCE**

**DEC 16, 2023**

# INTRODUCTION

In the age of digital communication, the surge in unwanted and potentially harmful SMS spam has become a significant concern with the exponential expansion of mobile communication and messaging services. Spam messages can vary from fraudulent schemes to irrelevant promotions which may lead to privacy intrusion or potential financial losses and cause an overall inconvenience to the user.

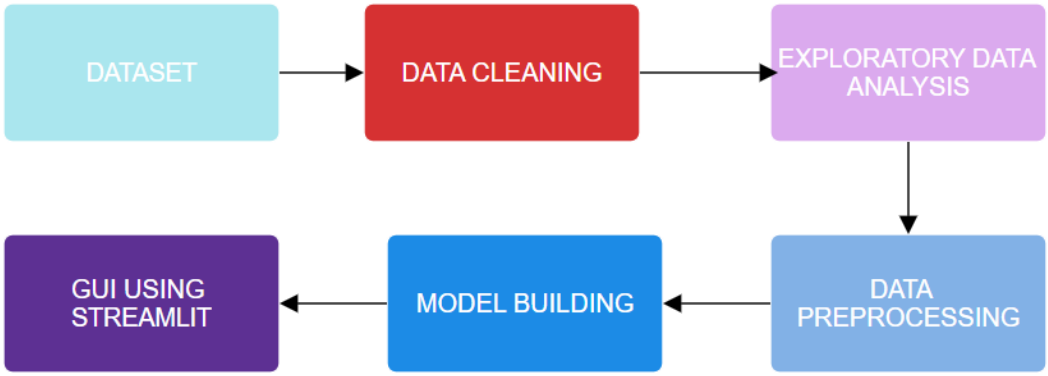
This issue can be addressed by building an effective SMS spam classifier that can differentiate between spam and non-spam messages in real-time, to ensure seamless and secure communication between users.

Machine learning techniques can be implemented to categorize messages as spam or not spam. In this report, a Multinomial Naïve Bayes model is used to develop a Streamlit-based application for classification purposes.

Naïve Bayes is a classification algorithm based on Bayes theorem. It consists of three types which include Gaussian, Multinomial, and Bernoulli Naïve Bayes. Gaussian Naïve Bayes is appropriate for continuous data and is used in classification tasks involving continuous variables as features. Multinomial Naive Bayes is used for classification where the data is discrete like text and consists of features that are word counts (occurrences) or term frequencies. In Bernoulli Naive Bayes, the features are assumed as binary variables.

The GUI is developed using Streamlit which is a Python library that is used for creating web applications. Streamlit enables the developer to convert static data scripts into interactive and captivating dashboards using minimal code. Data visualization libraries and machine learning frameworks are seamlessly integrated in Streamlit which makes it a powerful tool to bridge the divide between complex and user-friendly interfaces.

# FLOWCHART




**DATASET**

Dataset from Kaggle consists of about 5,574 messages, tagged according to ham or spam.

**DATA CLEANING**

Data cleaning involves fixing or eliminating inaccurate, corrupted, improperly formatted, duplicated, or incomplete data present within a dataset.

**EXPLORATORY DATA ANALYSIS**

Exploratory Data Analysis is a data analytics process to understand the data in depth and understand data characteristics using visualizations like pie chart, histograms, confusion matrix, word cloud etc.

**DATA PREPROCESSING**

Preprocessing here consists of changing the message into lowercase, tokenization, removing special characters(like $@& etc.), removing stop words(like is, of etc.) and punctuations and stemming.

**MODEL BUILDING**

Model is built using three types of Naive Bayes classifier i.e. Multinomial Naive Bayes, Gaussian Naive Bayes and Bernoulli Naive Bayes classifier. Accuracy, precision and confusion matrix for all three are checked.

**GUI USING STREAMLIT**

The trained model is saved and loaded in python code written for GUI. Using streamlit the website should take in SMS messages and classify based on the model.

# CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
from sklearn.naive_bayes import MultinomialNB
```

```
[ ] from wordcloud import WordCloud
    from collections import Counter
```

```
[ ] df = pd.read_csv('/content/spam.csv',encoding='latin-1')
    df.head()
```

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

```
[ ] df.shape
```

```
(5572, 5)
```

```
[ ] 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
```

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

```
[ ] df.sample(5)
```

| | v1 | v2 |
|---|---|---|
| 5272 | ham | Hello.How u doing?What u been up 2?When will u... |
| 4008 | ham | Ha... Then we must walk to everywhere... Canno... |
| 524 | spam | URGENT!: Your Mobile No. was awarded a å£2,000... |
| 2535 | ham | Can you pls pls send me a mail on all you know... |
| 4302 | ham | Yup i'm free... |

```
[ ] df.rename(columns={'v1':'target','v2':'text'},inplace=True)
    df.sample(5)
```

| | target | text |
|---|---|---|
| 3075 | ham | Mum, hope you are having a great day. Hoping t... |
| 4820 | ham | Im good! I have been thinking about you... |
| 5503 | ham | Perhaps * is much easy give your account ident... |
| 2024 | ham | U having lunch alone? I now so bored... |
| 4397 | ham | Can you tell Shola to please go to college of ... |

```
[ ] 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... |

```
[ ] df.isnull().sum()
```

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

```
[ ] df.duplicated().sum()
```

```
[ ] target    0
    text      0
    dtype: int64
```

```
[ ] df.duplicated().sum()
```

```
403
```

```
[ ] df = df.drop_duplicates(keep='first')
    df.duplicated().sum()
```

```
0
```

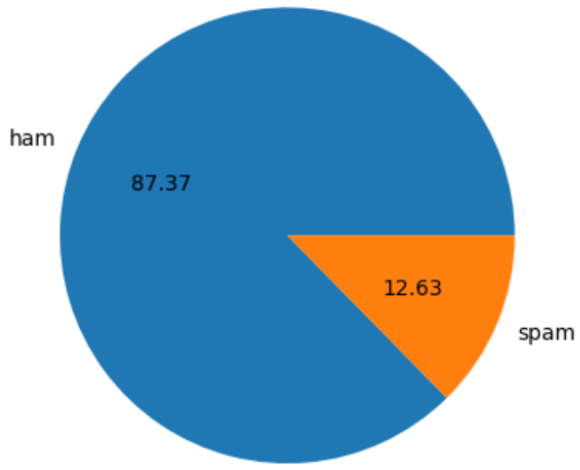
```
[ ] df.shape
```

```
(5169, 2)
```

```
[ ] df['target'].value_counts()
```

```
0    4516
1     653
Name: target, dtype: int64
```

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



```
[ ] !pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
```

```
[ ] import nltk
```

```
[ ] nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[ ] df['num_characters'] = df['text'].apply(len)
    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 |

```
[ ] df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
    df.head()
```

| | target | text | num_characters | num_words |
|---|---|---|---|---|
| **0** | 0 | Go until jurong point, crazy.. Available only ... | 111 | 24 |
| **1** | 0 | Ok lar... Joking wif u oni... | 29 | 8 |
| **2** | 1 | Free entry in 2 a wkly comp to win FA Cup fina... | 155 | 37 |
| **3** | 0 | U dun say so early hor... U c already then say... | 49 | 13 |
| **4** | 0 | Nah I don't think he goes to usf, he lives aro... | 61 | 15 |

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

| | target | text | num_characters | num_words | num_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 |

```
[ ] 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 |

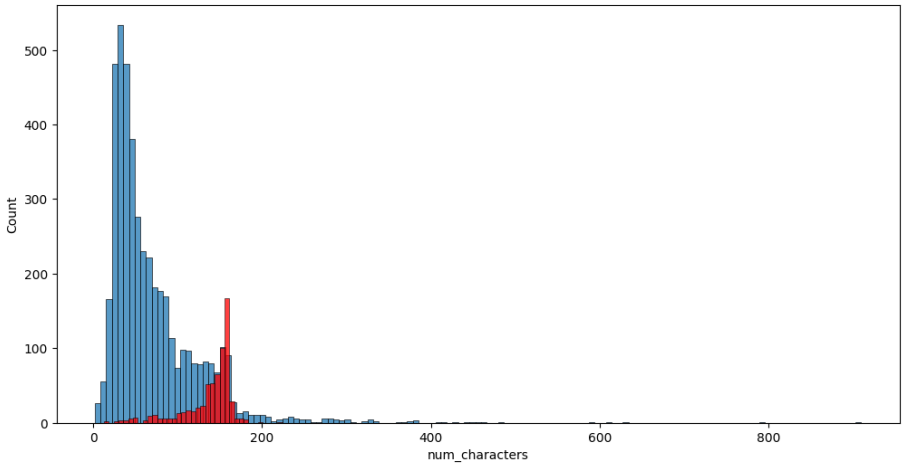
```
[ ] #for ham
    df[df['target'] == 0][['num_characters','num_words','num_sentences']].describe()
```

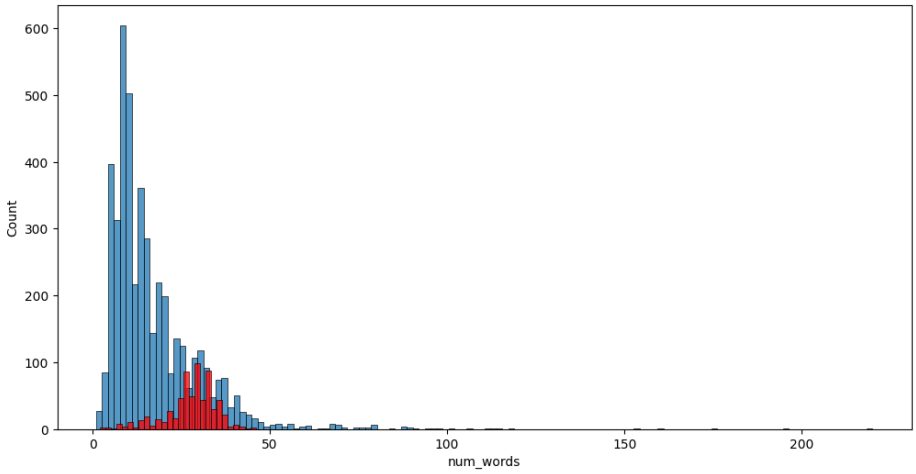
| | num_characters | num_words | num_sentences |
|---|---|---|---|
| count | 4516.000000 | 4516.000000 | 4516.000000 |
| mean | 70.459256 | 17.123782 | 1.820195 |
| std | 56.358207 | 13.493970 | 1.383657 |
| min | 2.000000 | 1.000000 | 1.000000 |
| 25% | 34.000000 | 8.000000 | 1.000000 |
| 50% | 52.000000 | 13.000000 | 1.000000 |
| 75% | 90.000000 | 22.000000 | 2.000000 |
| max | 910.000000 | 220.000000 | 38.000000 |

```
[ ] #for spam
    df[df['target'] == 1][['num_characters','num_words','num_sentences']].describe()
```

| | num_characters | num_words | num_sentences |
|---|---|---|---|
| count | 653.000000 | 653.000000 | 653.000000 |
| mean | 137.891271 | 27.667688 | 2.970904 |
| std | 30.137753 | 7.008418 | 1.488425 |
| min | 13.000000 | 2.000000 | 1.000000 |
| 25% | 132.000000 | 25.000000 | 2.000000 |
| 50% | 149.000000 | 29.000000 | 3.000000 |
| 75% | 157.000000 | 32.000000 | 4.000000 |
| max | 224.000000 | 46.000000 | 9.000000 |

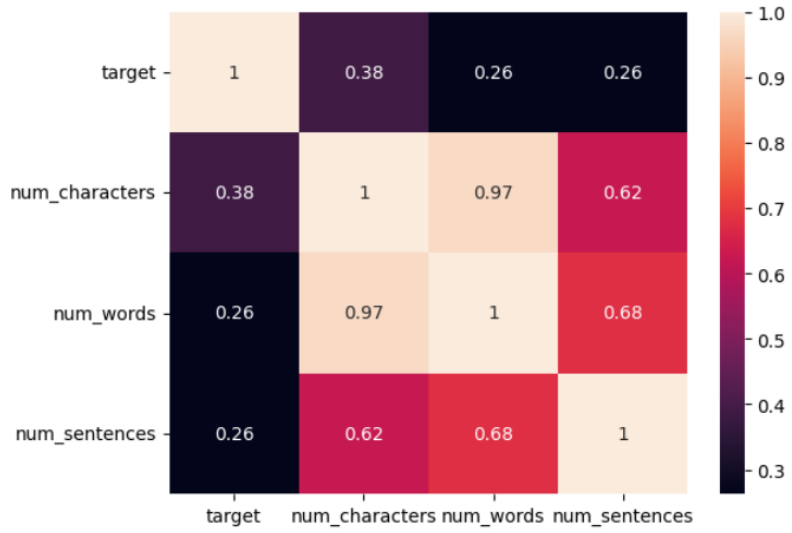
```
[ ] plt.figure(figsize=(12,6))
    sns.histplot(df[df['target'] == 0]['num_characters'])
    sns.histplot(df[df['target'] == 1]['num_characters'],color='red')

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




```
[ ]  sns.heatmap(df.corr(),annot=True)
```

```
<ipython-input-38-8df7bcac526d>:1: FutureWarning: The default value of numeric_only in DataFrame.cor
  sns.heatmap(df.corr(),annot=True)
<Axes: >
```



## DATA PREPROCESSING

```
[ ] import string
    from nltk.corpus import stopwords
```

```
[ ] nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
[ ] def transform_text(text):
        text = text.lower()
        text = nltk.word_tokenize(text)

        y = []
        for i in text:
            if i.isalnum():
                y.append(i)

        text = y[:]
        y.clear()

        for i in text:
            if i not in stopwords.words('english') and i not in string.punctuation:
                y.append(i)

        text = y[:]
        y.clear()

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


        return " ".join(y)
```

```
[ ] transform_text("Your gonna have to pick up a $1 burger for yourself on your way home. I can't even move. Pain is killing me.")
```

```
'gon na pick 1 burger way home ca even move pain kill'
```

```
[ ] df['text'][60]
```

```
'Your gonna have to pick up a $1 burger for yourself on your way home. I can't even move. Pain is killing me.'
```

```
[ ] from nltk.stem.porter import PorterStemmer
    ps = PorterStemmer()
    ps.stem('loving')
```

```
'love'
```

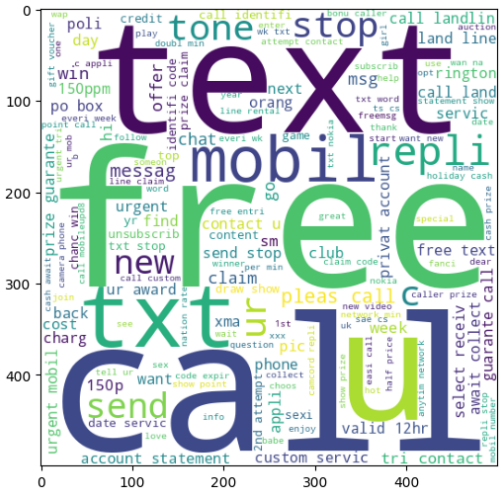
```
[ ] df['transformed_text'] = df['text'].apply(transform_text)
    df.head()
```

| | target | text | num_characters | num_words | num_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 |

```
[ ]  wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

```
[ ]  spam_wcld = wc.generate(df[df['target'] == 1]['transformed_text'].str.cat(sep=" "))
```

```
[ ]  plt.figure(figsize=(15,6))
     plt.imshow(spam_wcld)
```




```
[ ]  ham_wcld = wc.generate(df[df['target'] == 0]['transformed_text'].str.cat(sep=" "))
     plt.figure(figsize=(15,6))
     plt.imshow(ham_wcld)
```

```
[ ] df.head()
```

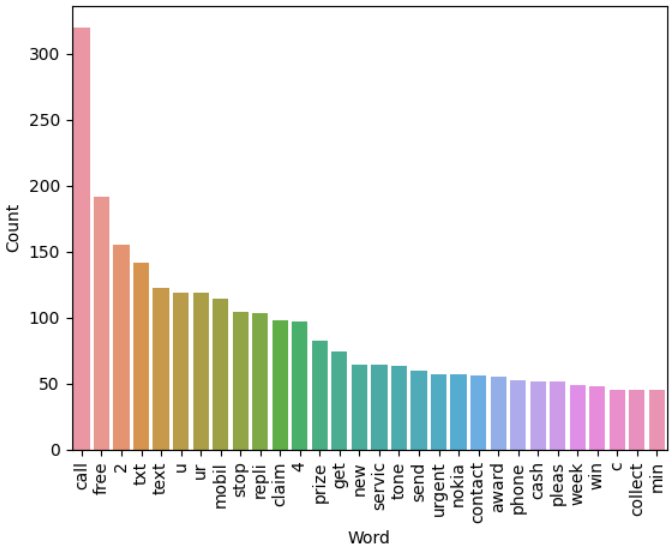
| | target | text | num_characters | num_words | num_sentences | transformed_text |
|---|---|---|---|---|---|---|
| 0 | 0 | Go until jurong point, crazy.. Available only ... | 111 | 24 | 2 | go jurong point crazi avail bugi n great world... |
| 1 | 0 | Ok lar... Joking wif u oni... | 29 | 8 | 2 | ok 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 |

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

```
[ ] len(spam_corpus)
```

```
9939
```

```
word_counts = Counter(spam_corpus).most_common(30)
df_word_counts = pd.DataFrame(word_counts, columns=['Word', 'Count'])

sns.barplot(x='Word', y='Count', data=df_word_counts)
plt.xticks(rotation='vertical')
plt.show()
```

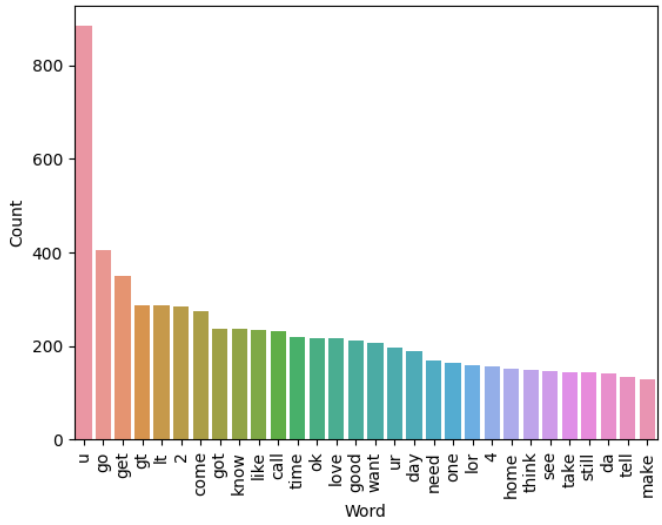


```
[ ] ham_corpus = []
    for msg in df[df['target'] == 0]['transformed_text'].tolist():
        for word in msg.split():
            ham_corpus.append(word)
```

```
[ ] len(ham_corpus)
```

```
35404
```

```
[ ] word_counts = Counter(ham_corpus).most_common(30)
    df_word_counts = pd.DataFrame(word_counts, columns=['Word', 'Count'])

    sns.barplot(x='Word', y='Count', data=df_word_counts)
    plt.xticks(rotation='vertical')
    plt.show()
```




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

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

```
[ ] X.shape
```

```
(5169, 3000)
```

```
[ ] y = df['target'].values
```

```
[ ] X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

```
[ ] gnb = GaussianNB()
    mnb = MultinomialNB()
    bnb = BernoulliNB()
```

```
[ ]  gnb.fit(X_train,y_train)
     y_pred1 = gnb.predict(X_test)
     print(f"Accuracy: {accuracy_score(y_test, y_pred1)}")
     print(confusion_matrix(y_test, y_pred1))
     print(f"Precision: {precision_score(y_test, y_pred1)}")
```

```
Accuracy: 0.8694390715667312
[[788 108]
 [ 27 111]]
Precision: 0.5068493150684932
```

```
[ ]  mnb.fit(X_train,y_train)
     y_pred2 = mnb.predict(X_test)
     print(f"Accuracy: {accuracy_score(y_test, y_pred2)}")
     print(confusion_matrix(y_test, y_pred2))
     print(f"Precision: {precision_score(y_test, y_pred2)}")
```

```
Accuracy: 0.9709864603481625
[[896   0]
 [ 30 108]]
Precision: 1.0
```

```
[ ]  bnb.fit(X_train,y_train)
     y_pred3 = bnb.predict(X_test)
     print(f"Accuracy: {accuracy_score(y_test, y_pred3)}")
     print(confusion_matrix(y_test, y_pred3))
     print(f"Precision: {precision_score(y_test, y_pred3)}")
```

```
Accuracy: 0.9835589941972921
[[895   1]
 [ 16 122]]
Precision: 0.991869918699187
```

```
[ ]  import pickle
     pickle.dump(tfidf,open('vectorizer.pkl','wb'))
     pickle.dump(mnb,open('model.pkl','wb'))
```

# GUI USING STREAMLIT

app.py | model.pkl | vectorizer.pkl

app.py > ...

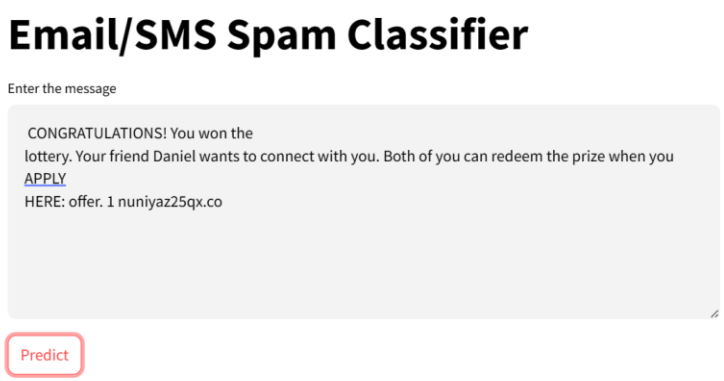
```python
import streamlit as st
import pickle
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

try:
    nltk.data.find('tokenizers/punkt')
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('punkt')
    nltk.download('stopwords')

def transform_text(text):
    text = text.lower()
    tokens = nltk.word_tokenize(text)

    tokens = [token for token in tokens if token.isalnum()]

    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words and token not in string.punctuation]


    tokens = [ps.stem(token) for token in tokens]

    return " ".join(tokens)

tfidf = pickle.load(open('vectorizer.pkl', 'rb'))
model = pickle.load(open('model.pkl', 'rb'))

st.title("Email/SMS Spam Classifier")

input_sms = st.text_area("Enter the message")

if st.button('Predict'):
    transformed_sms = transform_text(input_sms)
    vector_input = tfidf.transform([transformed_sms])
    result = model.predict(vector_input)[0]

    if result == 1:
        st.header("Spam")
    else:
        st.header("Not Spam")

```

# OUTPUT

Message 1:


Message 2:


Message 3:



# RESULTS AND CONCLUSION

The aim of this project was to classify whether a message is a spam message or not. It focuses on using three variations of the Naive Bayes Classifier, namely Gaussian, Multinomial, and Bernoulli. We used Python in Google Colab to build a model. Data cleaning and visualization techniques like heatmaps, word clouds, and graphs were employed for a comprehensive understanding of the dataset. From the word cloud we were able to view the commonly used words from the data in both Spam and Ham and the heatmap showed the correlation between the features of the dataset. From the heatmap, it could be inferred that the features 'num_words', 'num_characters' and 'num_sentences' had a high correlation and from the word cloud it was seen that the spam dataset had words such as "free", "text" and "call" commonly used while the ham dataset had words such as "come", "ok" and "time" as the commonly used words.

After data cleaning and visualization, the preprocessed dataset was split into testing and training data. The algorithms were then trained using the processed data, and their performance metrics, specifically accuracy and precision, were thoroughly compared. Bernoulli NB classifier had the highest accuracy of about 98% with multinomial NB ranking second with 97% and then Gaussian NB with 86%. Since we are dealing with a highly imbalanced dataset, we choose the algorithm with the highest precision which is Multinomial NB algorithm with a precision of 1 and accuracy of 97%. After this we built a user interface in VS code using streamlit where users can input a text message and test whether it is Spam or Not Spam. In Conclusion, we have successfully built an SMS Spam Classifier using Multinomial NB Algorithm.