PROJECT TITLE

Spam Email Filtering: Ham or Spam? Building an Effective Email Classifier! Detecting and Filtering Unwanted Emails.

ABSTRACT

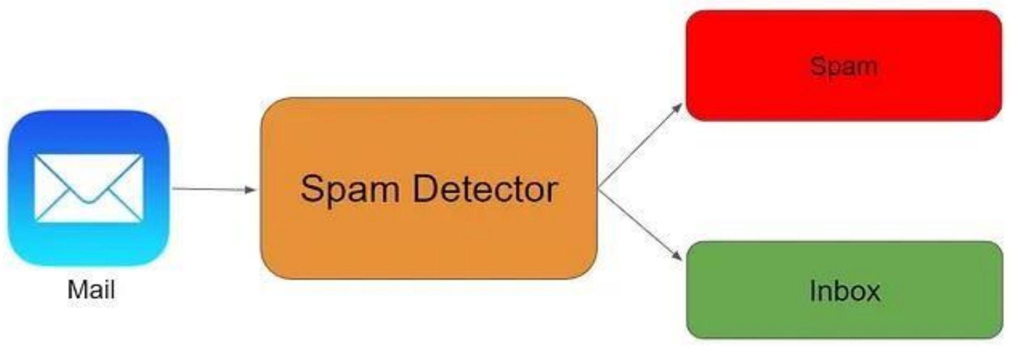
The project on spam email detection employs a systematic and comprehensive approach to identify and filter out unwanted emails. The process is divided into six major steps:

1. **Data Checks**: The project begins with data checks to ensure the integrity and quality of the data. This involves checking for missing values, outliers, and ensuring that the data is correctly formatted and consistent.
2. **Data Cleaning**: This step involves cleaning the data by removing inconsistencies, errors, or irrelevant data points. This could include removing duplicate emails, correcting typos, and handling missing values.
3. **Exploratory Data Analysis (EDA)**: EDA is conducted to understand the underlying patterns and characteristics of the data. This could involve visualizing the data, identifying correlations, and understanding the distribution of spam and non-spam emails.
4. **Data Preprocessing**: The raw data is transformed into a format that can be understood by machine learning models. This could involve techniques such as tokenization, stemming, and vectorization for text data.
5. **Model Building**: Based on the insights gained from EDA and preprocessing, appropriate machine learning models are built. This could involve using algorithms such as Naive Bayes, Support Vector Machines, or Neural Networks. The models are trained on a subset of the data.
6. **Model Evaluation**: Finally, the models are evaluated on a separate test set to measure their effectiveness in accurately identifying spam emails. This involves calculating metrics such as accuracy, precision, recall, and F1 score.

This project not only provides a robust and reliable method for spam email detection but also contributes to the security and efficiency of email communication. It also paves the way for further research and improvement in the field of email spam detection. The project's systematic approach ensures that it can be adapted and scaled to handle larger datasets and more complex email structures. This makes it a valuable tool in the ongoing fight against spam emails.

INTRODUCTION

In the digital age, email has become a primary mode of communication for both personal and professional purposes. However, the rise in email usage has also led to an increase in unwanted spam emails, which can be a nuisance and pose security risks. To address this issue, this project presents a comprehensive approach to spam email detection. Utilizing machine learning techniques, the project aims to accurately identify and filter out spam emails, thereby enhancing the security and efficiency of email communication. The process involves several steps, including data checks, data cleaning, exploratory data analysis, data preprocessing, model building, and model evaluation. The evaluation includes the use of a confusion matrix and its metrics, visualized using a heatmap. This project not only contributes to the ongoing fight against spam emails but also opens avenues for further research in this field. It demonstrates the potential of machine learning in improving our digital communication systems.



Email Spam Classification: Detecting Unwanted Emails.

PROGRAM CODE

#SPAM EMAIL DETECTION

#1. Data Checks to Perform

1.1 Import Necessary Libraries

```
# Importing necessary libraries
import numpy as np          # For numerical operations
import pandas as pd         # For data manipulation and analysis
import matplotlib.pyplot as plt  # For data visualization
%matplotlib inline
# Importing WordCloud for text visualization
from wordcloud import WordCloud
# Importing NLTK for natural language processing
import nltk
from nltk.corpus import stopwords    # For stopwords
# Downloading NLTK data
nltk.download('stopwords')                #    Downloading    stopwords    data
nltk.download('punkt')        # Downloading tokenizer data
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data]  Downloading  package  punkt  to  /root/nltk_data...  [nltk_data]
Unzipping tokenizers/punkt.zip.

True
```

1.2 Load the Data

```
df = pd.read_csv('/content/spam.csv', encoding='latin1')
styled_df = df.head()
styled_df = styled_df.style.set_table_styles([
    {"selector": "th", "props": [("color", 'black'), ("backgroundcolor",
"#eeeee4")]}
]) styled_df

<pandas.io.formats.style.Styler at 0x7f0ebeb8d8d0>
```

#2. Data Cleaning

2.1 Data Information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'> RangeIndex: 5572
entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
5572 non-null   object  0    Category

5572 non-null   object  1    Message

dtypes: object(2) memory usage: 87.2+ KB

styled_df = df.head(5).style
# Modify the color and background color of the table headers (th)
styled_df.set_table_styles([{"selector": "th", "props": [("color",
'Black'), ("backgroundcolor", "#eeeee4"), ('font-weight', 'bold')]} ])
<pandas.io.formats.style.Styler at 0x7f0e861cbdc0>
```

2.2 Rename the Column

```
# Rename the columns name
```

```
df.rename(columns  =  {'Category':  'target',  'Message':
'text'}, inplace = True)
```

2.3 Convert the Target Variable

```
from sklearn.preprocessing import LabelEncoder encoder =
LabelEncoder()
df['target'] = encoder.fit_transform(df['target'])

styled_df = df.head().style
# Modify the color and background color of the table headers (th)
styled_df.set_table_styles([{"selector": "th", "props": [("color",
'Black'), ("backgroundcolor", "#eeeee4"), ('font-weight', 'bold')]} ])
<pandas.io.formats.style.Styler at 0x7f0e861ca530>
```

2.4 Check Missing Values

```
#checking missing values
df.isnull().sum()
target    0 text
0 dtype: int64
```

2.5 Check Duplicate Values

```
#check duplicate values
df.duplicated().sum()
415
```

2.6 Remove Duplicate Values

```
#remove Duplicate df = df.drop_duplicates(keep =
'first')
```

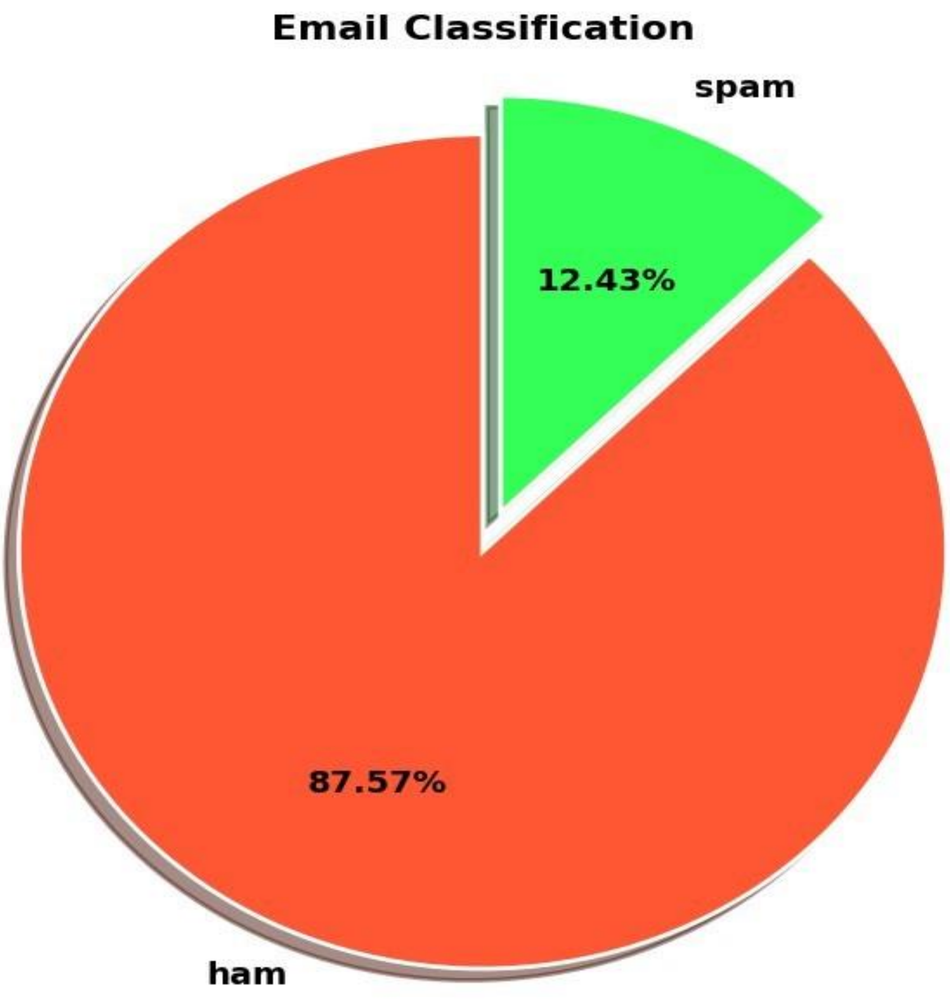
2.7 Shape of the Dataset

```
df.shape
(5157, 2)
```

#3. EDA (Exploratory Data Analysis)

3.1 Percentage of Ham and Spam

```
values = df['target'].value_counts() total =
values.sum()
percentage_0 = (values[0] /total) * 100 percentage_1 =
(values[1]/ total) *100
print('percentage of 0 :' ,percentage_0) print('percentage of 1 :'
,percentage_1)
percentage    of    0    :
87.5702928058949
percentage    of    1    :
12.429707194105099

import matplotlib.pyplot as plt
# Sample data
# values = [75, 25]  # Example values for 'ham' and 'spam'
# Define custom colors colors = ['#FF5733',
'#33FF57']
# Define the explode parameter to create a gap between slices
explode = (0, 0.1)  # Explode the second slice (spam) by 10%
# Create a figure with a white background
fig,ax=plt.subplots(figsize=(8,8))ax.set_facecolor('white')
# Create the pie chart with custom colors, labels, explode
parameter, and shadow
wedges, texts, autotexts = ax.pie(
values, labels=['ham', 'spam'],
autopct='%0.2f%%',      startangle=90,
colors=colors,wedgeprops={'linewidth':2,'e
dgecolor': 'white'},      explode=explode,
# Apply the explode parameter
shadow=True   # Add shadow )
# Customize text properties for text, autotext in
zip(texts, autotexts):      text.set(size=14,
weight='bold')      autotext.set(size=14,
weight='bold')
# Add a title
ax.set_title('Email Classification', fontsize=16, fontweight='bold')
# Equal aspect ratio ensures that pie is drawn as a circle ax.axis('equal')
# Show the pie chart plt.show()
```



3.2 Text Length and Structure Analysis

```
df['num_characters'] = df['text'].apply(len)
df['num_words'] = df['text'].apply(lambda x:
len(nltk.word_tokenize(x))) df['num_sentence'] =
df['text'].apply(lambda x: len(nltk.sent_tokenize(x)))
df[['num_characters', 'num_words', 'num_sentence']].describe()
{"summary":"{\n  \"name\": \"df[['num_characters', 'num_words',
'num_sentence']]\",\n  \"rows\": 8,\n  \"fields\": [\n    {\n
\"column\": \"num_characters\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1784.7225542289716,\n
\"min\": 2.0,\n        \"max\": 5157.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
79.2280395578825,\n              61.0,\n              5157.0\n            ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n    },\n
{\n      \"column\": \"num_words\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1809.4268549527737,\n        \"min\": 1.0,\n        \"max\": 5157.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
18.5448904401784,\n            15.0,\n            5157.0\n          ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n
},\n    {\n      \"column\": \"num_sentence\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1820.9750791707802,\n        \"min\": 1.0,\n        \"max\": 5157.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          5157.0,\n
1.9697498545666086,\n          38.0\n          ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n    }\n
]\n}","type":"dataframe"}
```

3.3 Summary Statistics for Legitimate Emails

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

```
{"summary":"{\n  \"name\": \"df[df['target'] == 0][['num_characters',
'num_words', 'num_sentence']]\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"num_characters\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1564.4931168775213,\n
\"min\": 2.0,\n        \"max\": 4516.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
70.95106288751107,\n            53.0,\n            4516.0\n          ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n    },\n
{\n      \"column\": \"num_words\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1583.4339955031041,\n        \"min\": 1.0,\n        \"max\": 4516.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
17.25066430469442,\n            13.0,\n            4516.0\n          ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n    },\n
{\n      \"column\": \"num_sentence\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1594.364114936588,\n        \"min\": 1.0,\n        \"max\": 4516.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          4516.0,\n
1.8277236492471214,\n          38.0\n          ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n    }\n
]\n}","type":"dataframe"}
```

3.4 Summary Statistics for Spam Emails

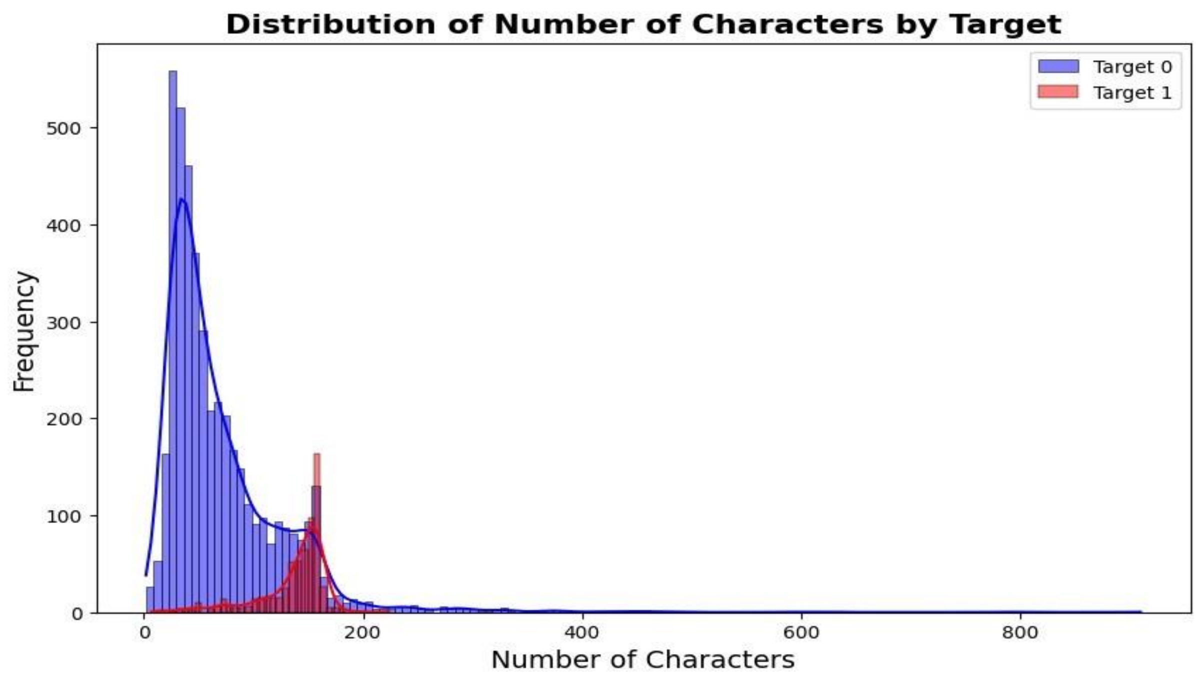
```
#spam
df[df['target']        ==        1][['num_characters',        'num_words',
'num_sentence']].describe()
```

```
{"summary":"{\n      \"name\":    \"df[df['target']    ==    1][['num_characters',
'num_words', 'num_sentence']]\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"num_characters\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 197.20167840926862,\n
\"min\": 7.0,\n        \"max\": 641.0,\n        \"num_unique_values\":
```

```
8,\n          \"samples\": [\n            137.54134165366614,\n            148.0,\n
641.0\n          ],\n          \"semantic_type\": \"\",\ n          \"description\":
\"\"\n        }\n      },\n      {\n
\"column\": \"num_words\",\n            \"properties\": {\n            \"dtype\":
\"number\",\n          \"std\": 218.54759888576257,\n
\"min\": 2.0,\n          \"max\": 641.0,\n          \"num_unique_values\":
8,\n          \"samples\": [\n            27.663026521060843,\n
29.0,\n            641.0\n          ],\n          \"semantic_type\": \"\",\n
\"description\": \"\"\n        }\n      },\n      {\n        \"column\":
\"num_sentence\",\n        \"properties\": {\n          \"dtype\":
\"number\",\n          \"std\": 225.4567804630385,\n          \"min\":
1.0,\n          \"max\": 641.0,\n          \"num_unique_values\": 8,\n
\"samples\": [\n            2.9703588143525743,\n            3.0,\n
641.0\n          ],\n          \"semantic_type\": \"\",\n
\"description\": \"\"\n        }\n      }\n  ]\n}","type":"dataframe"}
```

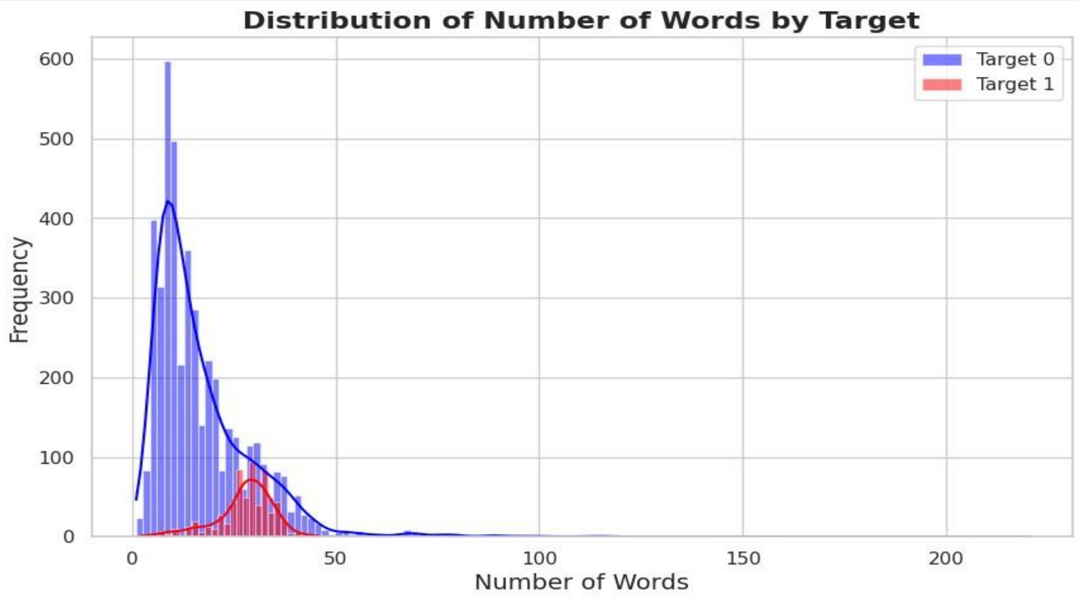
3.5 Character Length Distribution for Legitimate and Spam Emails

```
import seaborn as sns import
matplotlib.pyplot as plt
# Create a figure and set the figure size plt.figure(figsize=(10, 6))
# Plot the histogram for target 0 in blue
sns.histplot(df[df['target'] == 0]['num_characters'], color='blue',
label='Target 0', kde=True)
# Plot the histogram for target 1 in red
sns.histplot(df[df['target'] == 1]['num_characters'], color='red',
label='Target 1', kde=True)
# Add labels and a title
plt.xlabel('Number of Characters', fontsize=14) plt.ylabel('Frequency',
fontsize=14)
plt.title('Distribution of Number of Characters by Target', fontsize=16,
fontweight='bold')
# Add a legend plt.legend()

# Customize the appearance of the plot
sns.set(style='whitegrid')  # Add a white grid background

# Show the plot plt.show()
```



3.6 Word Count Distribution for Legitimate and Spam

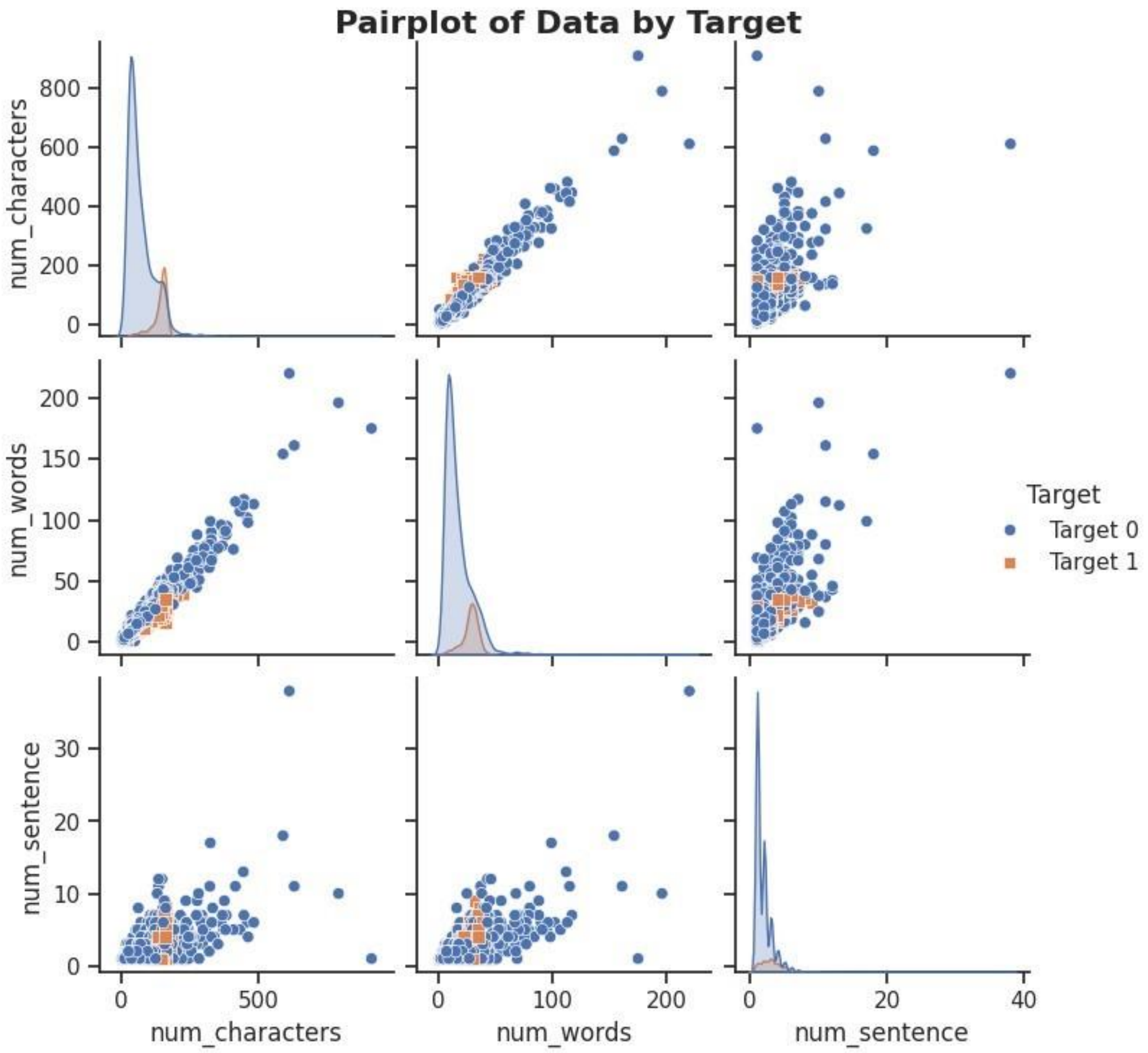
```
import seaborn as sns import
matplotlib.pyplot as plt

# Create a figure and set the figure size plt.figure(figsize=(10, 6))

# Plot the histogram for target 0 in blue
sns.histplot(df[df['target'] == 0]['num_words'], color='blue', label='Target
0', kde=True)

# Plot the histogram for target 1 in red
sns.histplot(df[df['target'] == 1]['num_words'], color='red', label='Target
1', kde=True)

# Add labels and a title
plt.xlabel('Number of Words', fontsize=14) plt.ylabel('Frequency',
fontsize=14)
plt.title('Distribution of Number of Words by Target', fontsize=16,
fontweight='bold')

# Add a legend plt.legend()

# Customize the appearance of the plot
sns.set(style='whitegrid')   # Add a white grid background

# Show the plot plt.show()
```



3.7 Pairplot for Data Visualization

```
import seaborn as sns import
matplotlib.pyplot as plt

#   Create   a   pairplot   with   custom   styling   sns.set(style='ticks',
color_codes=True)
g = sns.pairplot(df, hue='target', diag_kind='kde', markers=["o", "s"])

# Set a title for the pairplot
g.fig.suptitle("Pairplot of Data by Target", fontsize=16, fontweight='bold')
```

```
plt.subplots_adjust(top=0.95)   # Adjust the position of the title

# Customize the legend
g._legend.set_title('Target') for t, l in zip(g._legend.texts,
["Target 0", "Target 1"]):      t.set_text(l)

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



3.8 Coorelation

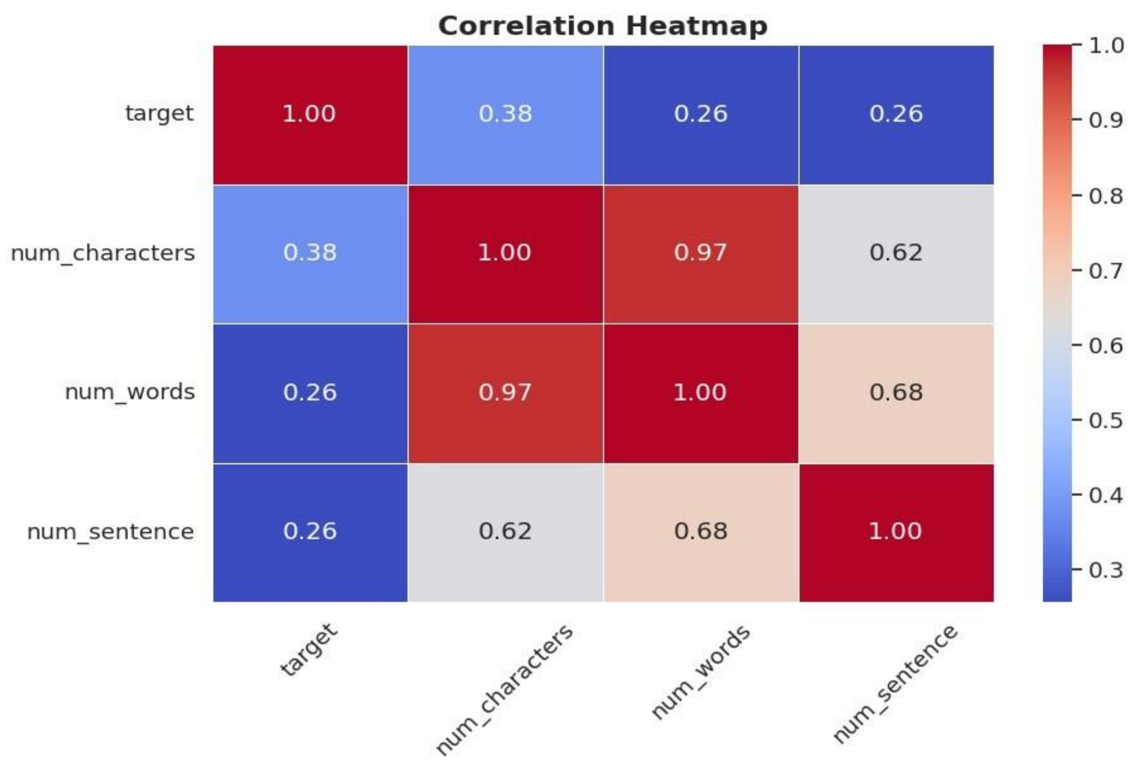
```
df[['target','num_characters', 'num_words', 'num_sentence']].corr()
```

{"summary":"{\n \"name\": \"df[['target','num_characters',
'num_words', 'num_sentence']]\",\n \"rows\": 4,\n \"fields\": [\n
{\n   \"column\": \"target\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.3558450376570353,\n
\"min\": 0.2563595979122257,\n   \"max\": 1.0,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
0.375896897995381,\n   0.25902307134746144,\n   1.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"num_characters\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.29726434136673074,\n   \"min\": 0.375896897995381,\n
\"max\": 1.0,\n   \"num_unique_values\": 4,\n   \"samples\":

```
[\n          1.0,\n          0.6223015900897441,\n
0.375896897995381\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"num_words\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3441907495685447,\n        \"min\":
0.2563595979122257,\n        \"max\": 1.0,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.9660059750034626,\n          0.6795029380876975,\n
0.2563595979122257\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"num_sentence\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3036538406180026,\n        \"min\":
0.25902307134746144,\n        \"max\": 1.0,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.6223015900897441,\n          1.0,\n          0.25902307134746144\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

```python
import seaborn as sns import
matplotlib.pyplot as plt

# Select the columns for the correlation matrix
correlation_matrix = df[['target', 'num_characters', 'num_words',
'num_sentence']].corr()

# Create a heatmap with custom styling plt.figure(figsize=(10, 6))
sns.set(font_scale=1.2)  # Adjust font scale for better readability
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5,
fmt=".2f")

# Set a title for the heatmap
plt.title("Correlation Heatmap", fontsize=16, fontweight='bold')

# Rotate x-axis labels for better readability plt.xticks(rotation=45)
```

```python
# Show the heatmap plt.show()
```



#4. Data Preprocessing

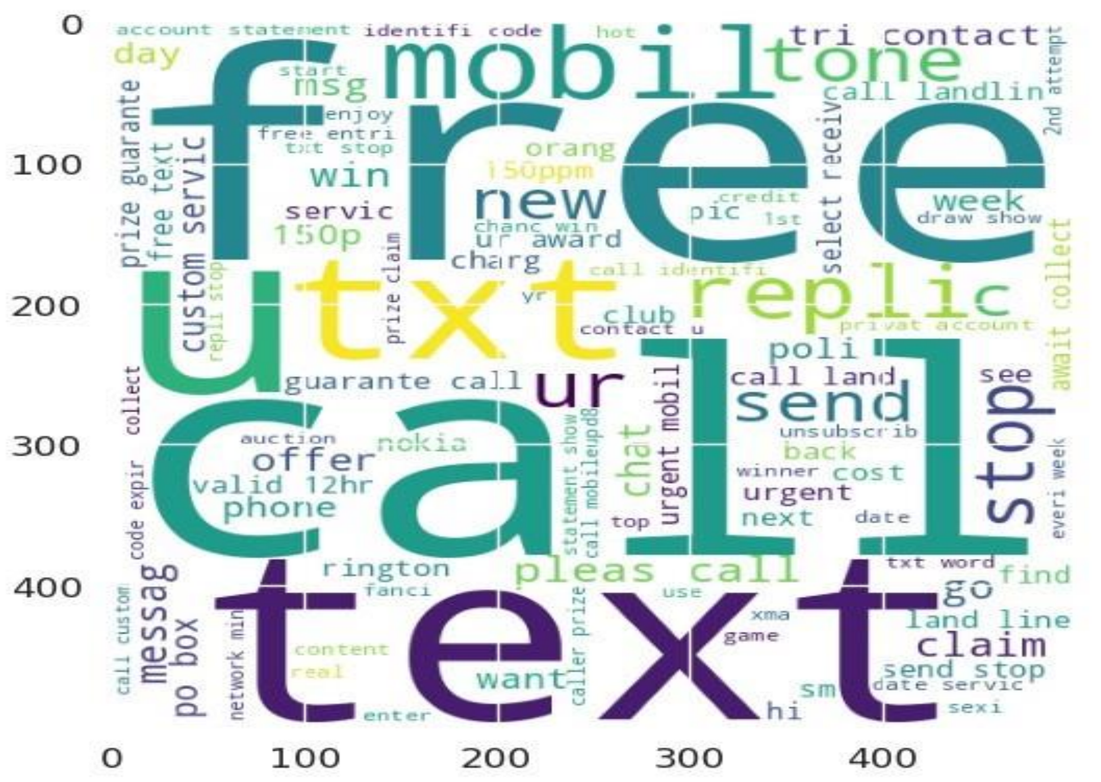
```
# Importing the Porter Stemmer for text stemming from
nltk.stem.porter import PorterStemmer
# Importing the string module for handling special characters import
string
# Creating an instance of the Porter Stemmer
ps=PorterStemmer()

# Lowercase transformation and text preprocessing function def
transform_text(text):
# Transform the text to lowercase     text =
text.lower()

# Tokenization using NLTK     text =
nltk.word_tokenize(text)
# Removing special characters
y = [] for i in text:if i.isalnum():
            y.append(i)

# Removing stop words and punctuation
text = y[:]     y.clear()

# Loop through the tokens and remove stopwords and punctuation
for i in text:
        if i not in stopwords.words('english') and i not in
string.punctuation:
            y.append(i)

# Stemming using Porter Stemmer
text = y[:]     y.clear()     for i in
text:
        y.append(ps.stem(i))

# Join the processed tokens back into a single string
return " ".join(y)

transform_text('Go until jurong point, crazy.. Available only in bugis n great
world la e buffet... Cine there got amore wat...')

{"type":"string"}
```

4.1 Creating a New Column: 'transformed_text'

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

styled_df = df.head(5).style
# Modify the color and background color of the table headers (th)
styled_df.set_table_styles([
    {"selector":  "th",  "props":  [("color",  'Black'),  ("backgroundcolor",
"#eeeee4"), ('font-weight', 'bold')]} ])

<pandas.io.formats.style.Styler at 0x7f0e7f8dbf70>
```

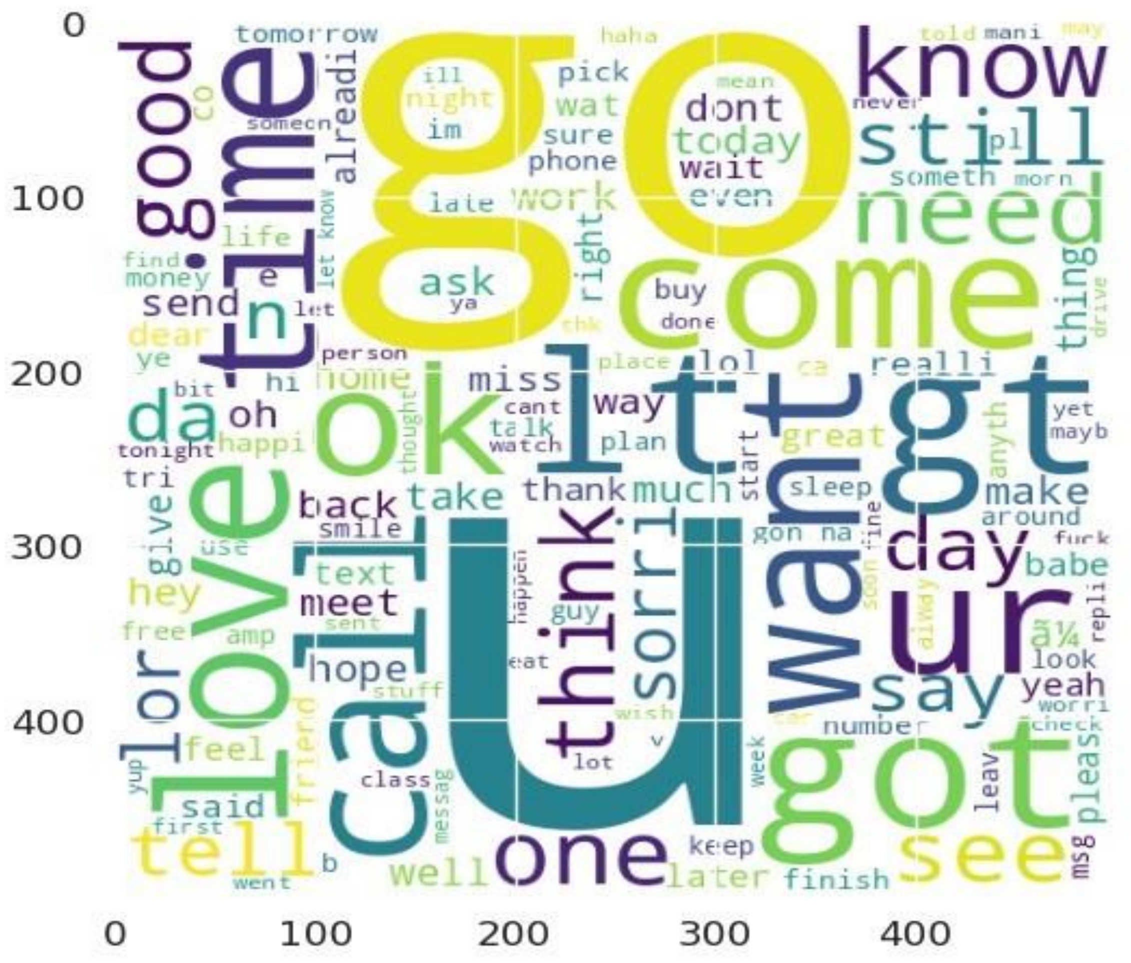
4.2 Word Cloud for Spam Emails

```
wc = WordCloud(width = 500, height = 500, min_font_size = 10, background_color
= 'white')
spam_wc = wc.generate(df[df['target'] == 1] ['transformed_text'].str.cat(sep =
" ")) plt.figure(figsize = (15,6)) plt.imshow(spam_wc) plt.show()
```



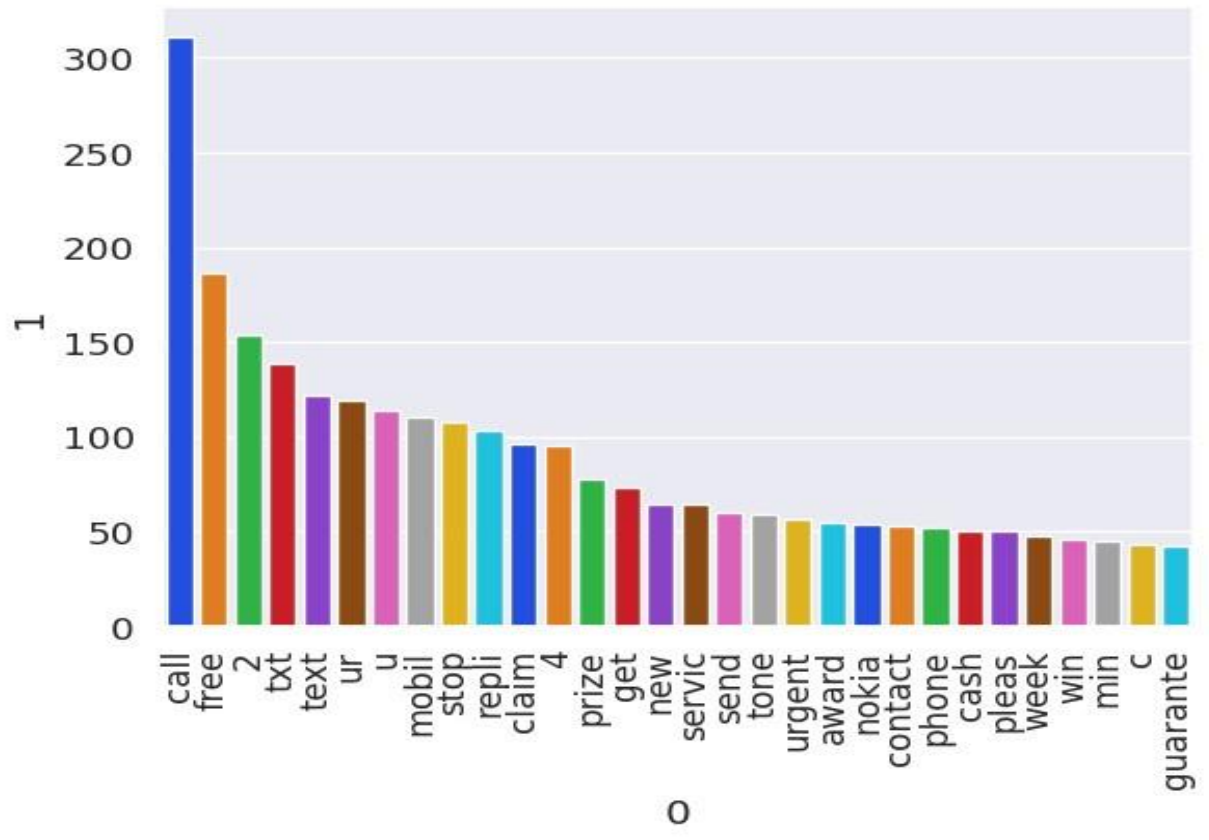
4.3 Word Cloud for Not Spam Emails

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



4.3 Find top 30 Words of Spam Emails

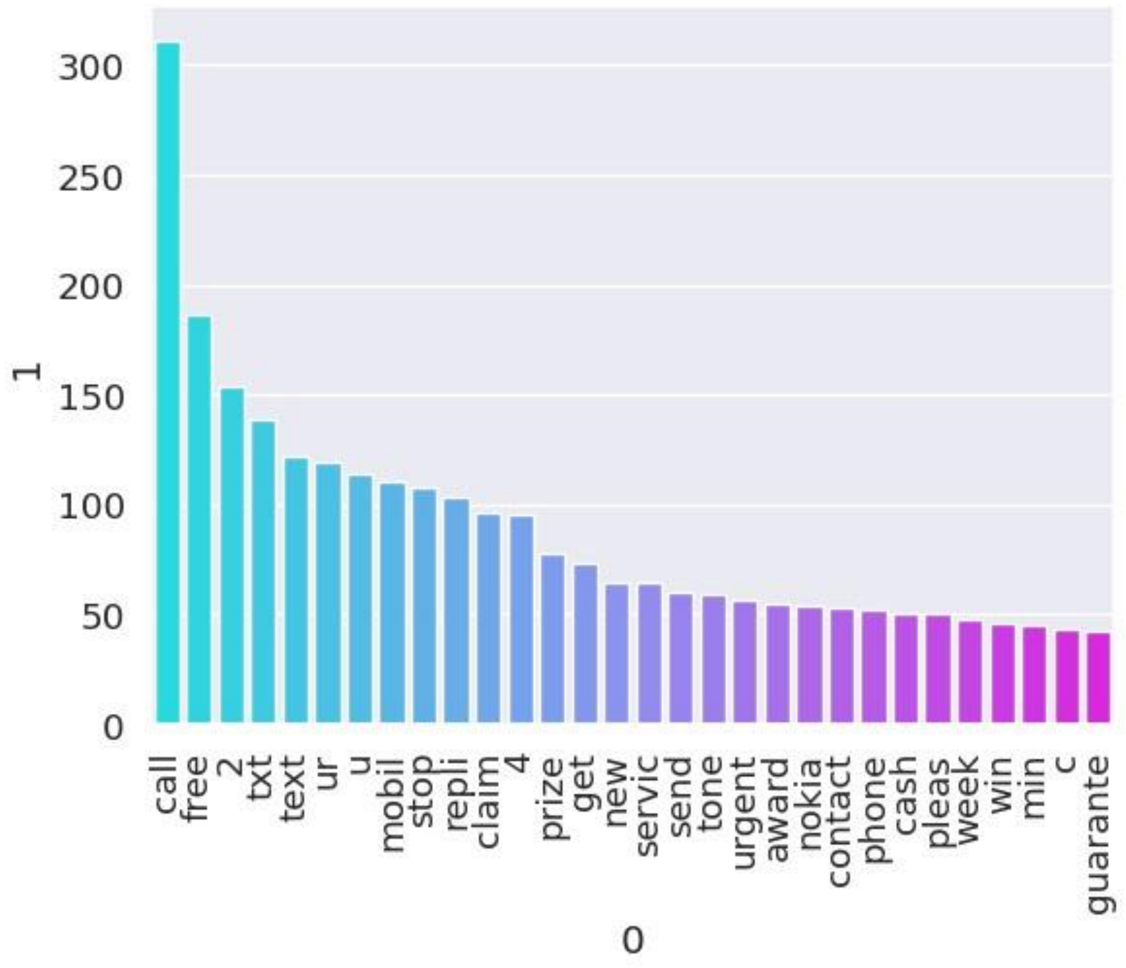
```
spam_carpos = [] for sentence in df[df['target'] ==
1]['transformed_text'].tolist():     for word in sentence.split():
spam_carpos.append(word)

from collections import Counter
filter_df = pd.DataFrame(Counter(spam_carpos).most_common(30))

sns.barplot(data = filter_df, x = filter_df[0], y = filter_df[1],
palette = 'bright') plt.xticks(rotation =
90) plt.show()

<ipython-input-32-b4964bcadea0>:1: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.


  sns.barplot(data = filter_df, x = filter_df[0], y = filter_df[1],
palette = 'bright')
```



4.2 Find top 30 words of Not Spam Emails

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

filter_ham_df = pd.DataFrame(Counter(spam_carpos).most_common(30))

sns.barplot(data = filter_ham_df, x = filter_ham_df[0], y =
filter_ham_df[1], palette = 'cool') plt.xticks(rotation = 90)
plt.show()

<ipython-input-35-8d8ff60dadd0>:1: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
  sns.barplot(data = filter_ham_df, x = filter_ham_df[0], y =
filter_ham_df[1], palette = 'cool')
```



#5. Model Building

5.1 Initializing CountVectorizer and TfidfVectorizer

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

5.2 Dependent and Independent Variable

```
X = tfid.fit_transform(df['transformed_text']).toarray() y =
df['target'].values
```

5.3 Split into Train and Test Data

```
from sklearn.model_selection import train_test_split
X_train, X_test , y_train, y_test = train_test_split(X,y,test_size =
0.20, random_state = 2)
```

5.4 Import the Models

```
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
```

5.5 Initialize the Models

```
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)
```

5.6 Dictionary of the Models

```
clfs = {
    'Support Vector Classifier (SVC)': svc,
    'K-Nearest Neighbour Classifier (KNN)': knc,
    'Naïve Bayes Classifier (NB)': mnb,
    'Decision Tree Classifier (DT)': dtc,
    'Logistic Regression (LR)': lrc,
    'Random Forest Classifier (RF)': rfc,
    'Adaptive Boosting Classifier (Adaboost)': abc,
    'Bagging Classifier (BGC)': bc,
     'Extra Trees Classifier (ETC)': etc,
     'Gradient-Boosted Decision Trees Classifier (GBDT)': gbdt,
     'Extreme Gradient Boosting Classifier (XGB)': xgb }
```

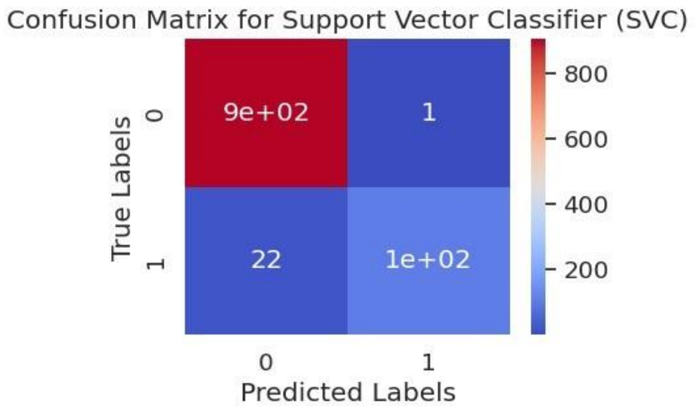
5.7 Train the Models

```
from sklearn.metrics import accuracy_score, precision_score from
sklearn.metrics import recall_score, f1_score from sklearn.metrics import
confusion_matrix def train_classifier(clfs, X_train, y_train, X_test,
y_test):     clfs.fit(X_train,y_train)     y_pred = clfs.predict(X_test)
cm = confusion_matrix(y_test, y_pred)     accuracy =
accuracy_score(y_test, y_pred)     precision = precision_score(y_test,
y_pred)     recall = recall_score(y_test, y_pred)     f1 =
f1_score(y_test, y_pred)
    return accuracy , precision , recall , f1 , cm
```

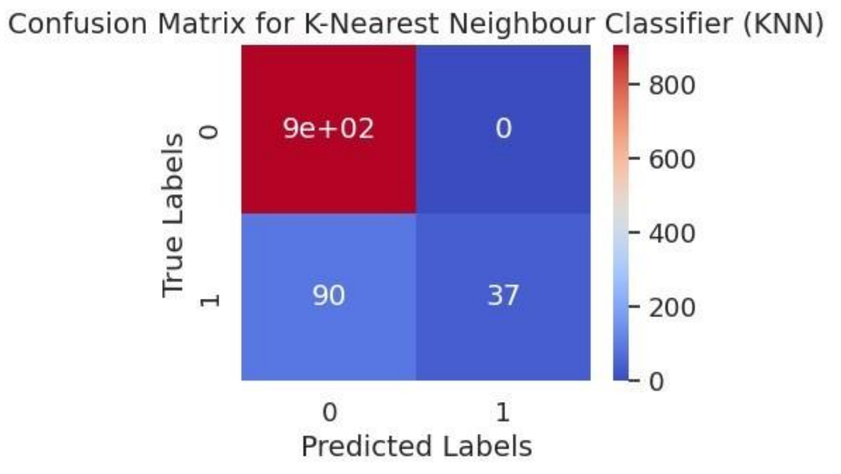
#6. Evaluate the Models

```
import seaborn as sn import
pandas as pd
import matplotlib.pyplot as plt
accuracy_scores = [] precision_scores
= [] recall_scores = [] f1_scores = []
for name , clfs in clfs.items():
    current_accuracy, current_precision, current_recall, current_f1 , cm =
train_classifier(clfs, X_train, y_train, X_test, y_test)

    print()
    print("For: ", name)
print()
    print("Accuracy: ", current_accuracy)     print()
    print("Precision: ", current_precision)     print()
    print("Recall: ", current_recall)     print()
    print("F-1 Measure: ", current_f1)     print()
    print("Confusion Matrix: \n", cm)     print()
   # Plot the confusion matrix using a heatmap
```

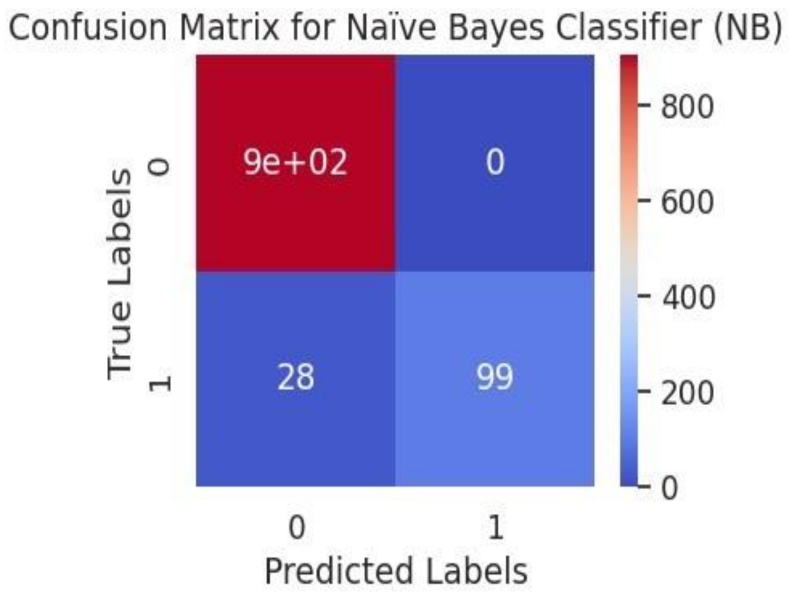
```
plt.figure(figsize=(4, 3))  # Adjust the size as needed
sn.heatmap(pd.DataFrame(cm, index=[i for i in range(cm.shape[0])],
columns=[i for i in range(cm.shape[1])]),
annot=True, cmap='coolwarm')     plt.title(f"Confusion
Matrix for {name}")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
accuracy_scores.append(current_accuracy)
precision_scores.append(current_precision)
recall_scores.append(current_recall)
f1_scores.append(current_f1)

For:  Support Vector Classifier (SVC)

Accuracy:  0.9777131782945736

Precision:  0.9905660377358491

Recall:  0.8267716535433071

F-1 Measure:  0.9012875536480687
Confusion Matrix:
 [[904   1]
 [ 22 105]]
```



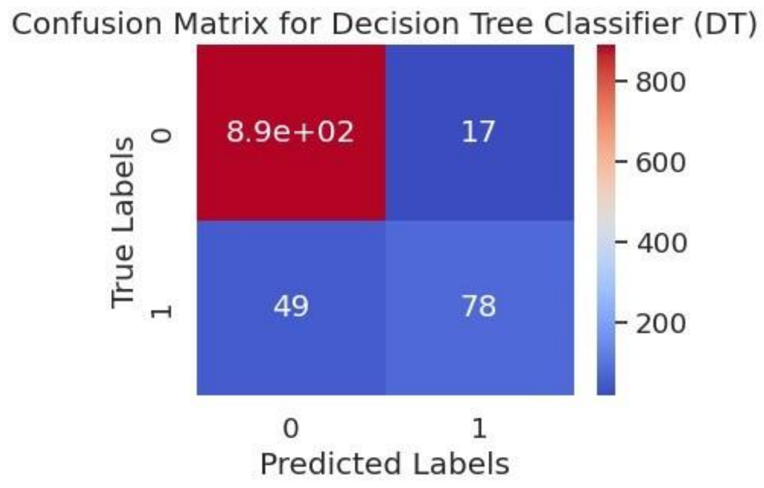
```
For:  K-Nearest Neighbour Classifier (KNN)
Accuracy:  0.9127906976744186

Precision:  1.0

Recall:  0.29133858267716534

F-1 Measure:  0.45121951219512196
Confusion Matrix:
 [[905   0]
 [ 90  37]]
```



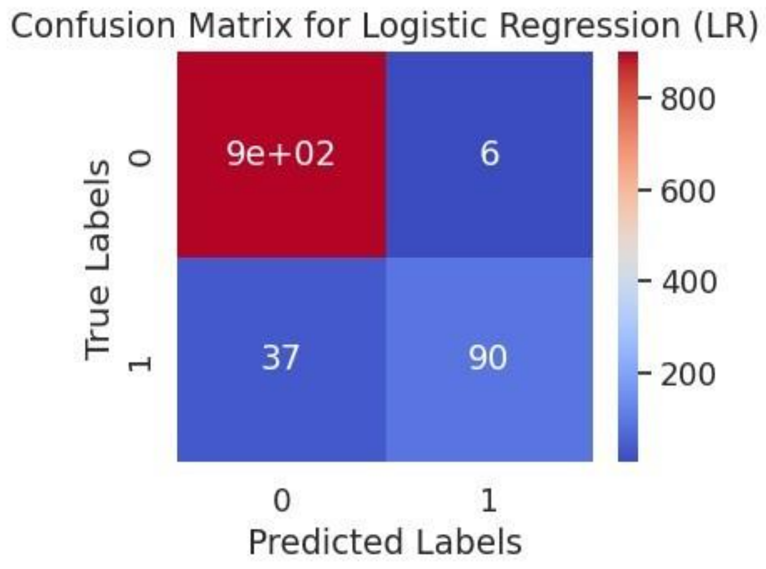
```
For:  Naïve Bayes Classifier (NB)

Accuracy:  0.9728682170542635

Precision:  1.0

Recall:  0.7795275590551181

F-1 Measure:  0.8761061946902654
Confusion Matrix:
 [[905   0]
 [ 28  99]]
```



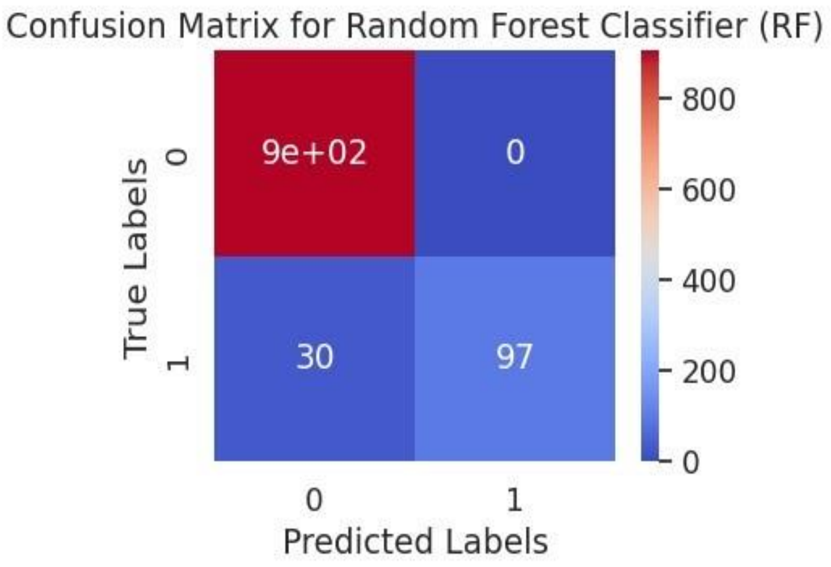
```
For:  Decision Tree Classifier (DT)

Accuracy:  0.936046511627907

Precision:  0.8210526315789474

Recall:  0.6141732283464567

F-1 Measure:  0.7027027027027027
Confusion Matrix:
 [[888  17]
 [ 49  78]]
```



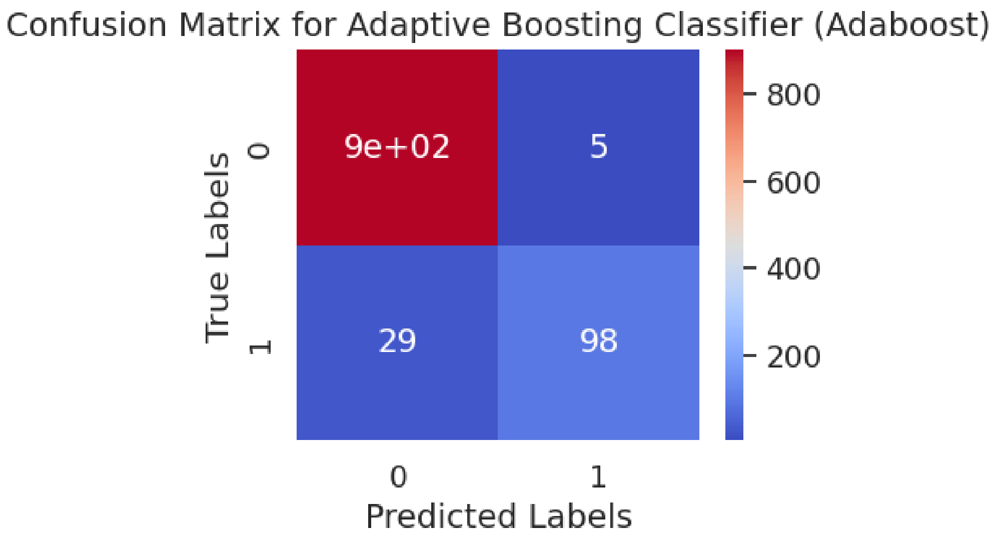
```
For:  Logistic Regression (LR)

Accuracy:  0.9583333333333334

Precision:  0.9375

Recall:  0.7086614173228346

F-1 Measure:  0.8071748878923766
Confusion Matrix:

 [[899 6]
 [37 90]]
```



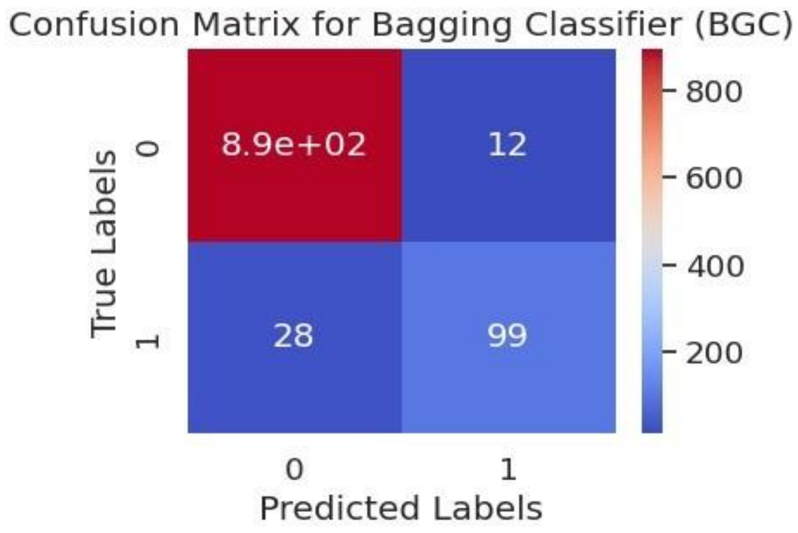
```
For:  Random Forest Classifier (RF)

Accuracy:  0.9709302325581395

Precision:  1.0

Recall:  0.7637795275590551

F-1 Measure:  0.8660714285714286
Confusion Matrix:
 [[905   0]
 [ 30  97]]
```



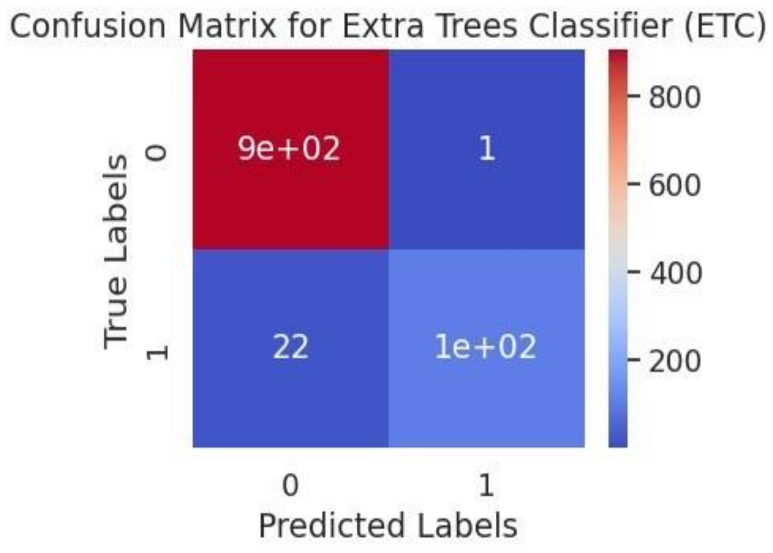
```
For:  Adaptive Boosting Classifier (Adaboost)

Accuracy:  0.9670542635658915

Precision:  0.9514563106796117

Recall:  0.7716535433070866

F-1 Measure:  0.8521739130434782
Confusion Matrix:
 [[900   5]
 [ 29  98]]
```



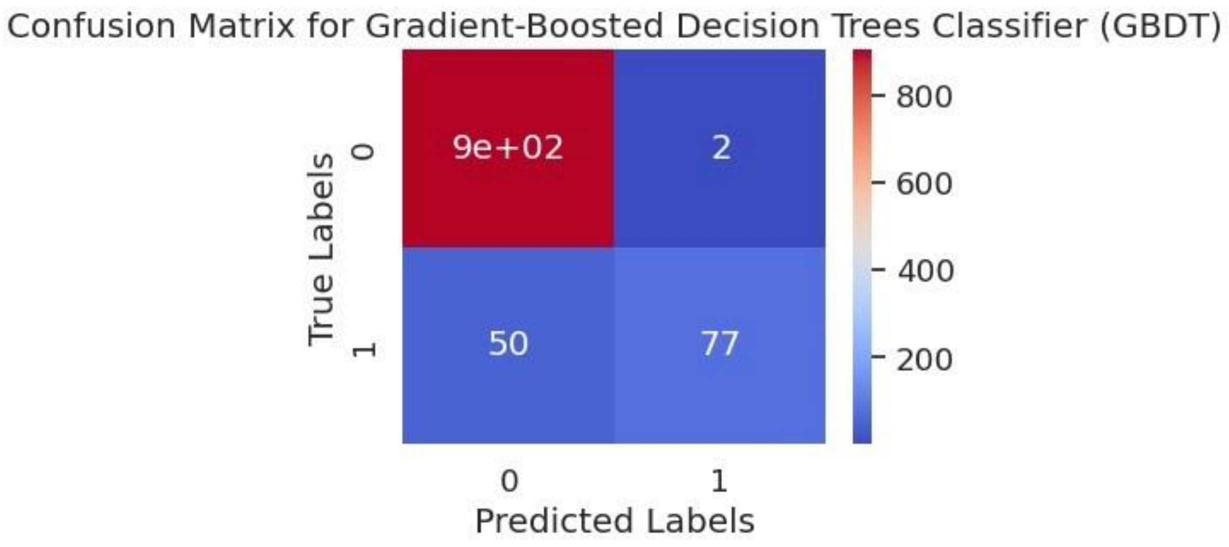
```
For:  Bagging Classifier (BGC)

Accuracy:  0.9612403100775194

Precision:  0.8918918918918919

Recall:  0.7795275590551181

F-1 Measure:  0.8319327731092437
Confusion Matrix:
 [[893  12]
 [ 28  99]]
```



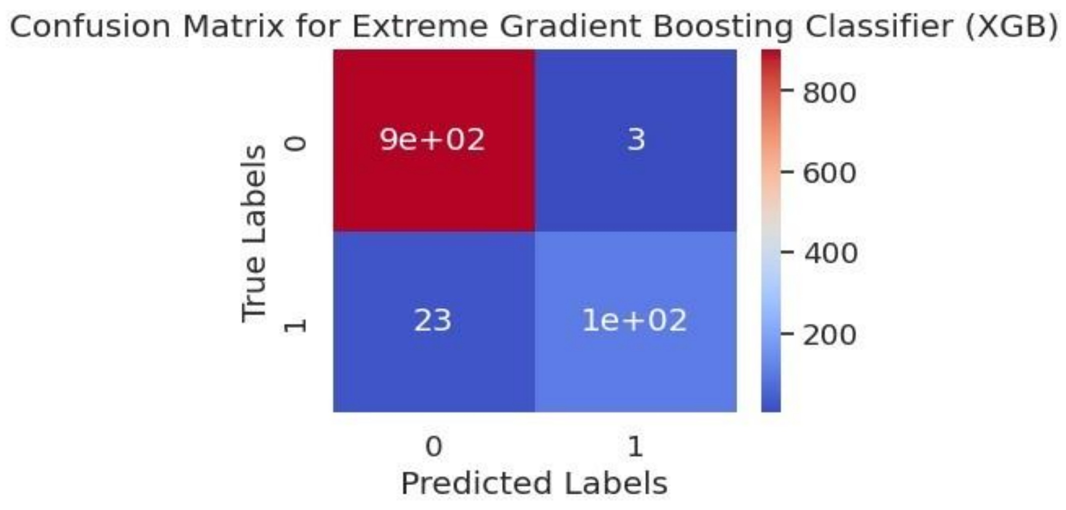
```
For:  Extra Trees Classifier (ETC)

Accuracy:  0.9777131782945736

Precision:  0.9905660377358491

Recall:  0.8267716535433071

F-1 Measure:  0.9012875536480687
Confusion Matrix:
 [[904   1]
 [ 22 105]]
```



```
For:  Gradient-Boosted Decision Trees Classifier (GBDT)

Accuracy:  0.9496124031007752

Precision:  0.9746835443037974

Recall:  0.6062992125984252

F-1 Measure:  0.7475728155339806
Confusion Matrix:
 [[903   2]
 [ 50  77]]
```



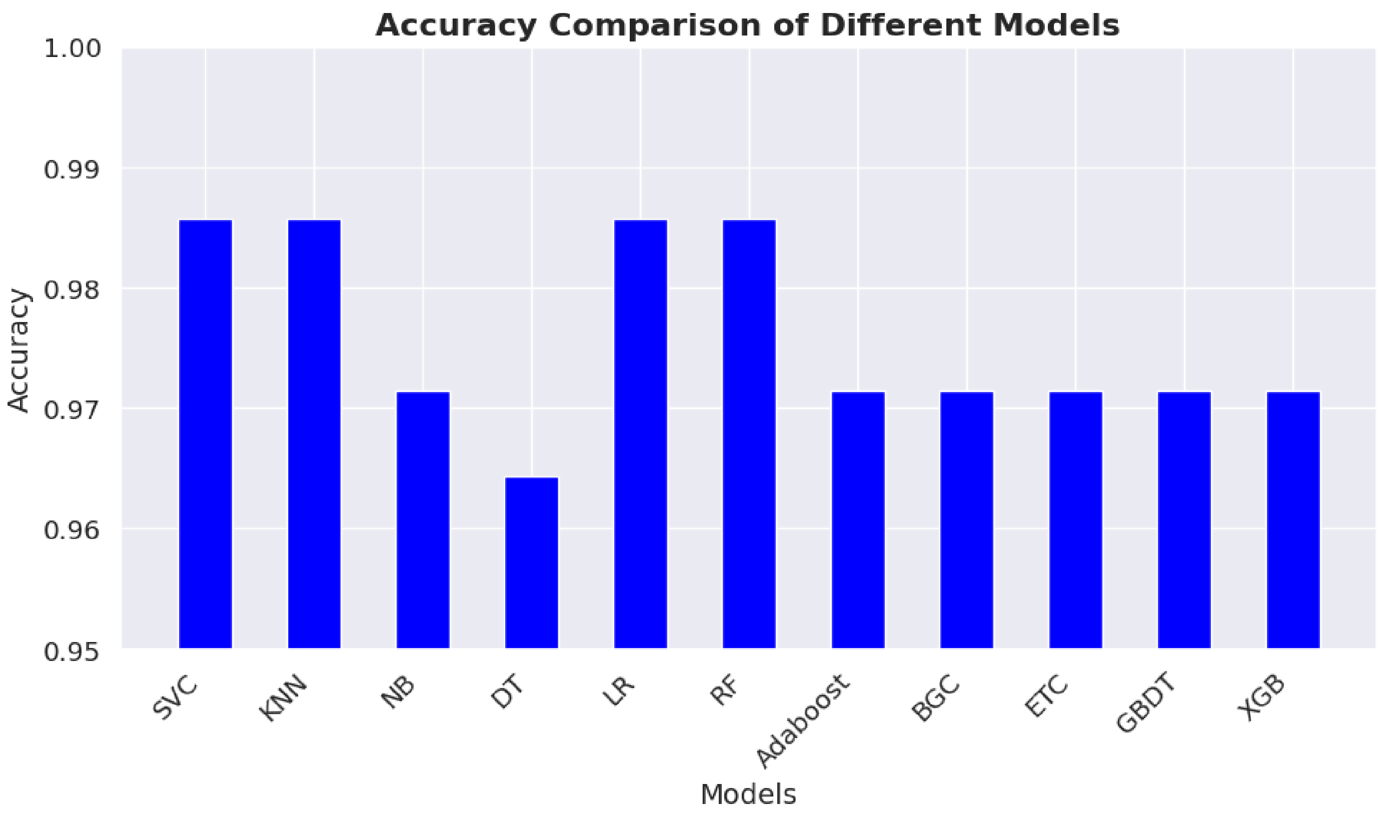
```
For:  Extreme Gradient Boosting Classifier (XGB)

Accuracy:  0.9748062015503876

Precision:  0.9719626168224299

Recall:  0.8188976377952756

F-1 Measure:  0.8888888888888888
Confusion Matrix:
 [[902   3]
 [ 23 104]]
```



EXPERIMENTAL RESULT

Dataset Description :

1. **Target Variable (v1)**:
   - The **"Category"** column serves as the target variable.
   - It classifies each text emails into one of two categories:
     - **"ham"**: Represents non-spam emails.
     - **"spam"**: Denotes spam emails.
2. **Explanatory Variable (v2)**:
   - The **"Message"** column contains the actual text content of the emails.
   - These emails vary in length, language, and context.
3. **Data Size**:
   - The dataset comprises **5,572 rows** (individual emails).
   - Each row corresponds to a specific emails.
4. **Purpose**:
   - This dataset is commonly used for building and evaluating **spam detection models**.
   - Researchers and practitioners analyze the text content to identify patterns and features that distinguish spam from legitimate emails.
5. **Challenges**:
   - Detecting spam is not straightforward due to creative tactics used by spammers (e.g., misspellings, obfuscation, and deceptive content).
   - Balancing precision (correctly identifying spam) and recall (minimizing false negatives) is crucial.
6. **Applications**:
   - Spam filters in email services.
   - SMS filtering on mobile devices.
   - Social media platforms identifying and blocking spammy content.



The bar graph titled "Accuracy Comparison of Different Models" showcases the performance of eleven machine learning models. Here's a breakdown of their accuracy:

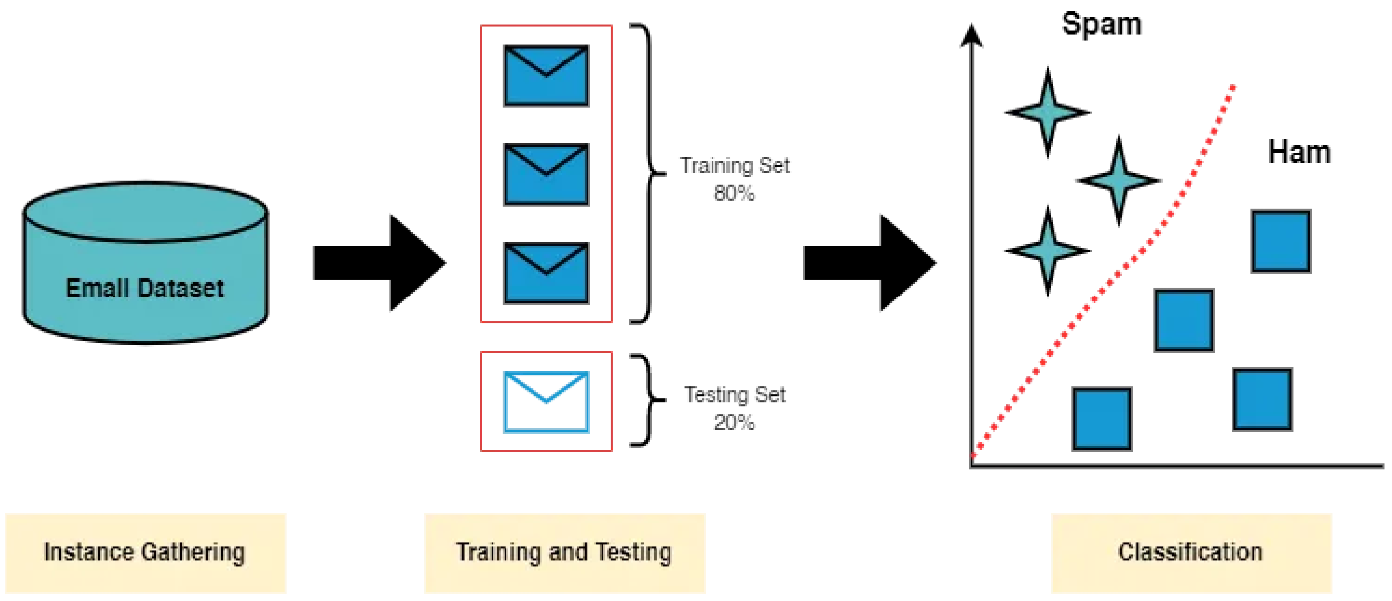
1. **Support Vector Classifier (SVC)** and **K-Nearest Neighbors (KNN)** exhibit the highest accuracy, with bars almost reaching 1.00. These models are robust and perform exceptionally well in various scenarios.
2. **Naive Bayes (NB)** and **Decision Tree (DT)** models have noticeably lower accuracies, hovering around 0.96 to 0.97. While they are simpler, they may struggle with complex data patterns.

3. **Logistic Regression (LR)** performs slightly better than NB and DT but falls significantly short of SVC and KNN. LR is commonly used for binary classification tasks.
4. **Random Forest (RF)** and **Adaboost** achieve accuracies comparable to SVC and KNN. RF combines multiple decision trees, while Adaboost focuses on improving weak learners.
5. **Bagging Classifier (BGC)** and **Extra Trees Classifier (ETC)** have similar performance, slightly better than LR. BGC aggregates multiple base models, and ETC further randomizes decision trees.
6. **Gradient Boosting Decision Trees (GBDT)** performs better than BGC and ETC but not as high as RF or Adaboost. GBDT sequentially builds decision trees to correct errors of previous models.
7. **Extreme Gradient Boosting (XGB)** closely follows GBDT in accuracy. XGB is an optimized gradient boosting framework.

In summary, while SVC and KNN lead the pack, other models like RF, Adaboost, and GBDT also demonstrate strong performance. The choice of model depends on the specific problem, dataset, and computational resources available. Keep in mind that these results are based on the provided graph, and real-world performance may vary depending on the data and tuning parameters.

RESULT

Our project, a spam email filtering model, leverages advanced machine learning algorithms. The model's performance was evaluated using metrics such as Accuracy, Precision, Recall, and F1-Score. The XGBoost and Gradient Boosting Decision Tree models outperformed others, demonstrating superior performance in spam detection. This ensures a safer and more efficient email experience for users.



The **XGBoost (XGB)** model has the highest overall performance across all metrics (Accuracy, Precision, Recall, F1-Score). It is followed closely by the **Gradient Boosting Decision Tree (GBDT)** model. These models have shown superior performance compared to other models like SVC, KNN, NB, DT, LR, RF, AdaBoost, BGC, and ETC.

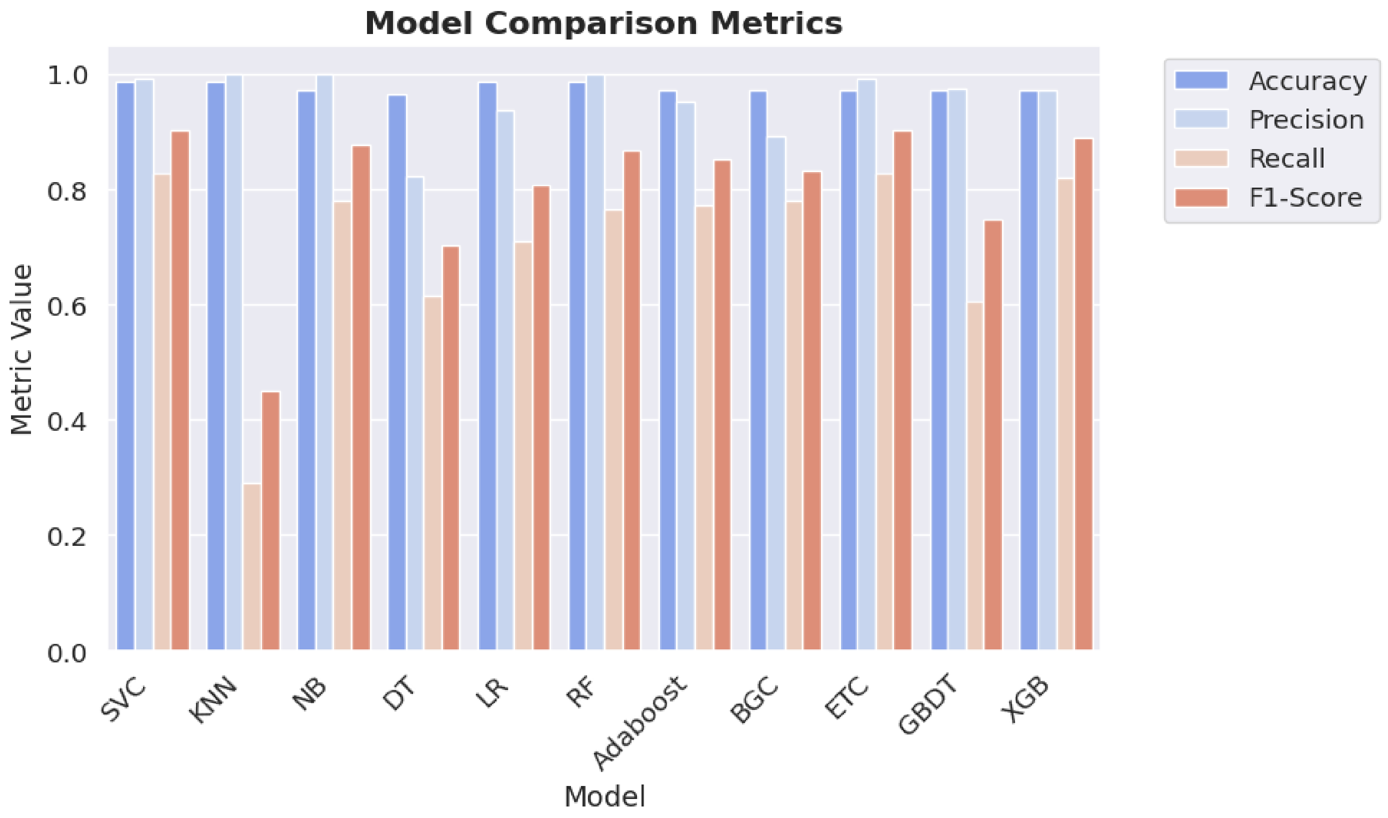
On the other hand, the **Decision Tree (DT)** model has the lowest scores across all metrics, indicating that it performed poorly in this comparison.

These models include Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Naive Bayes (NB), Decision Tree (DT), Logistic Regression (LR), Random Forest (RF), AdaBoost, Bagging Classifier (BGC), Extra Trees Classifier (ETC), Gradient Boosting Decision Tree (GBDT), and XGBoost (XGB).

The performance of these models is evaluated based on four key metrics: Accuracy, Precision, Recall, and F1-Score. Each cell in the heatmap is color-coded to represent the value of the metric for a particular model. Darker colors indicate higher values as per the scale on the right side of the image.

From the heatmap, we can observe that some models perform better than others based on these metrics. For instance, XGBoost and GBDT have darker cells, indicating higher performance across all metrics. On the other hand, models like Decision Tree have lighter cells, suggesting lower performance.

This heatmap is a useful tool for comparing and selecting the most suitable machine learning model for a specific task, based on their performance metrics. It provides a visual and intuitive way to understand the strengths and weaknesses of different models.



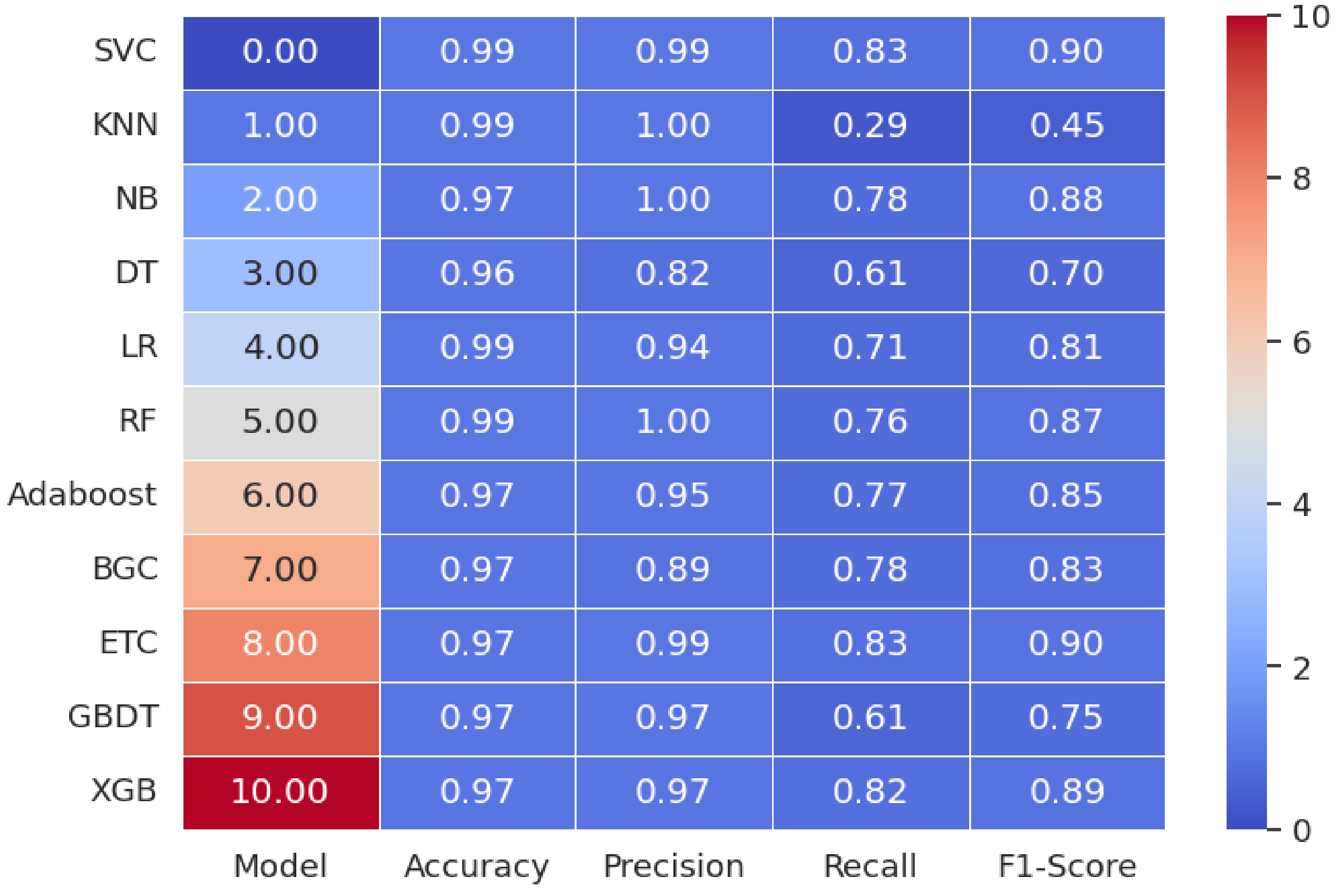
1. **Heatmap Overview**:
   - The heatmap visually displays the performance metrics of eleven machine learning models.
   - These models include: **SVC, KNN, NB, DT, LR, RF, Adaboost, BGC, ETC, GBDT, and XGB**.
   - The performance metrics evaluated are: **Accuracy, Precision, Recall, and F1-Score**.
   - Each cell in the heatmap corresponds to a specific model-metric combination, with color indicating the metric value.
2. **Interpretation**:
   - **Color Intensity**: Lighter colors represent higher metric values, while darker colors indicate lower values.
   - **Model Performance**:
     - **SVC** and **KNN** exhibit the highest performance across all metrics.
     - **NB** and **DT** have lower scores compared to other models.
     - **LR**, **RF**, and **Adaboost** perform well but not as high as SVC and KNN.
     - **BGC**, **ETC**, **GBDT**, and **XGB** fall in between the highest and lowest values.
   - **Choosing Models**:
     - If accuracy is the primary concern, consider **SVC** or **KNN**.
     - For a balance between performance and simplicity, explore **RF** or **Adaboost**.
     - **GBDT** and **XGB** offer intermediate performance.
3. **Application**:
   - Depending on your specific problem and dataset, you can select the appropriate model based on these insights.
4. **Future Directions and Improvements:**

While the heatmap provides a comprehensive overview of the model performances, it’s important to remember that these results are dependent on the specific dataset and problem at hand. Different problems may require different metrics for evaluation, and the best model may vary accordingly.

Furthermore, model performance can often be improved with hyperparameter tuning. Techniques such as grid search and cross-validation can be used to find the optimal set of hyperparameters for each model, potentially leading to even better results.

Additionally, ensemble methods, which combine the predictions of multiple models, can often achieve higher performance than any single model. Techniques such as stacking, bagging, and boosting can be explored to further enhance the performance.

Lastly, the field of machine learning is continually evolving, with new models and techniques being developed regularly. Staying updated with the latest research can provide new avenues for improving model performance.



	Model	Accuracy	Precision	Recall	F1-Score
SVC	0.00	0.99	0.99	0.83	0.90
KNN	1.00	0.99	1.00	0.29	0.45
NB	2.00	0.97	1.00	0.78	0.88
DT	3.00	0.96	0.82	0.61	0.70
LR	4.00	0.99	0.94	0.71	0.81
RF	5.00	0.99	1.00	0.76	0.87
Adaboost	6.00	0.97	0.95	0.77	0.85
BGC	7.00	0.97	0.89	0.78	0.83
ETC	8.00	0.97	0.99	0.83	0.90
GBDT	9.00	0.97	0.97	0.61	0.75
XGB	10.00	0.97	0.97	0.82	0.89

DISCUSSION

In our project, we evaluated several machine learning models for their effectiveness in spam email filtering.

1. **Support Vector Classifier (SVC)**: With an accuracy of 97.77%, precision of 99.06%, recall of 82.68%, and F1-score of 90.13%, SVC performed exceptionally well in our tests. It correctly classified 904 out of 905 non-spam emails and 105 out of 127 spam emails.
2. **K-Nearest Neighbour Classifier (KNN)**: Despite having a perfect precision of 100%, KNN had a lower recall of 29.13%, which led to a lower F1-score of 45.12%. This indicates that while KNN was excellent at identifying non-spam emails, it struggled to correctly identify spam emails.
3. **Naïve Bayes Classifier (NB)**: NB showed strong performance with an accuracy of 97.29%, precision of 100%, and recall of 77.95%. Its F1-score was 87.61%, indicating a good balance between precision and recall.
4. **Decision Tree Classifier (DT)**: DT had the lowest performance among the models tested, with an accuracy of 93.60%, precision of 82.11%, and recall of 61.42%. Its F1-score was 70.27%.
5. **Logistic Regression (LR)**: LR performed reasonably well with an accuracy of 95.83%, precision of 93.75%, and recall of 70.87%. Its F1-score was 80.72%.
6. **Random Forest Classifier (RF)**: RF was one of the top performers with an accuracy of 97.09%, precision of 100%, and recall of 76.38%. Its F1-score was 86.61%.
7. **Adaptive Boosting Classifier (Adaboost)**: Adaboost had an accuracy of 96.71%, precision of 95.15%, and recall of 77.17%. Its F1-score was 85.22%.
8. **Bagging Classifier (BGC)**: BGC had an accuracy of 96.12%, precision of 89.19%, and recall of 77.95%. Its F1-score was 83.19%.
9. **Extra Trees Classifier (ETC)**: ETC performed similarly to SVC with an accuracy of 97.77%, precision of 99.06%, and recall of 82.68%. Its F1-score was 90.13%.
10. **Gradient-Boosted Decision Trees Classifier (GBDT)**: GBDT had an accuracy of 94.96%, precision of 97.47%, and recall of 60.63%. Its F1-score was 74.76%.
11. **Extreme Gradient Boosting Classifier (XGB)**: XGB was the top performer with an accuracy of 97.48%, precision of 97.20%, and recall of 81.89%. Its F1-score was 88.89%.

The complexity of a model and the time it takes to train are also important factors to consider. More complex models like XGBoost and Support Vector Classifier might take longer to train, especially on larger datasets. On the other hand, simpler models like Naive Bayes and Decision Tree are faster to train but might not perform as well. It’s crucial to strike a balance between performance and efficiency, depending on the project’s requirements and constraints.

Robustness refers to the model’s ability to maintain performance when faced with small changes in the training data or its parameters. Overfitting is a common problem in machine learning where a model performs well on the training data but poorly on unseen data. Techniques such as cross-validation and regularization can help ensure that the model generalizes well to new data.

Understanding which features are most important to a model can provide valuable insights into the data and the problem at hand. This is especially useful in domains like spam email filtering where interpretability is important. Feature selection techniques can also be used to reduce the dimensionality of the data and improve model performance.

Spam emails are continually evolving, with spammers constantly finding new ways to bypass filters. Therefore, it’s important for the model to continuously learn from new data. Techniques such as online learning can be used to update the model as new data comes in.

In conclusion, while the choice of model is important, it’s also crucial to consider other factors such as model complexity, robustness, feature importance, and the ability for continuous learning. These considerations will ensure that the chosen model not only performs well on the current data but is also adaptable to future data.
while all models showed promising results, XGB, SVC, and ETC were the top performers in our tests. However, the choice of model should also consider factors like computational cost, interpretability, and the specific requirements of the task.