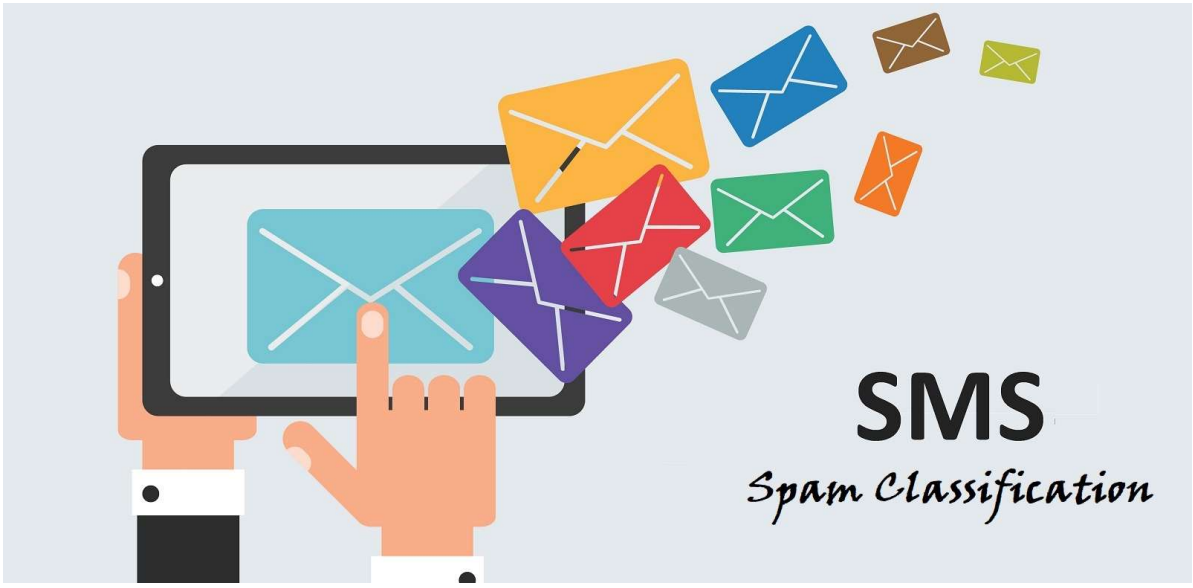
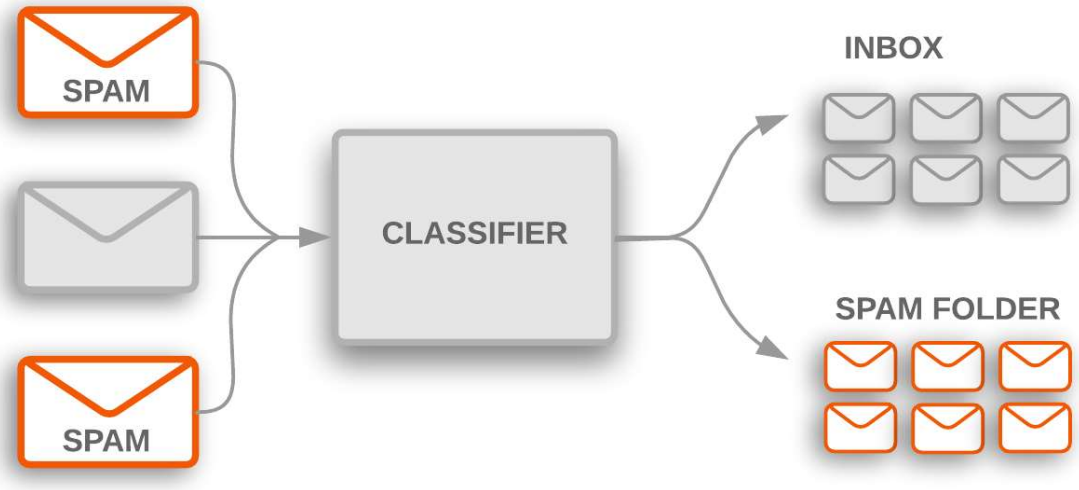


SMS Spam Collection Dataset

The SMS Spam Collection Dataset is a widely used dataset in machine learning and natural language processing (NLP) for text classification and spam detection tasks. It consists of a collection of SMS (Short Message Service) messages, where each message is labeled as either "ham" (legitimate) or "spam" (unsolicited or promotional). This dataset is valuable for building and evaluating spam detection algorithms and models.

SMS Spam Classification: Detecting Unwanted Messages



Life Cycle of the Project

Steps to be Performed Introduction Problem Statement 🙋 Data Checks to Perform 🌏 Data Cleaning 🧹 EDA 📊 Text Preprocessing ✍️ Model Training 🤖 Evaluation 📈 Conclusion 🎉 Author Message ✉️

1. Introduction

The SMS Spam Collection Dataset is a curated collection of SMS messages labeled as either "ham" (legitimate) or "spam" (unsolicited or promotional). It serves as a crucial resource for training and evaluating machine learning models in the field of spam detection and text classification. With origins in research, this dataset helps combat spam, aids in natural language processing (NLP) research, and enhances cybersecurity efforts by providing real-world examples of text messages for analysis and model development.

2. Problem Statement

In this project, we will be building an email spam detector using Python. Spam mail, also known as junk mail, refers to emails sent to a large number of users simultaneously, often containing deceptive messages, fraudulent schemes, or potentially harmful phishing content. We will utilize machine learning techniques to train our spam detector to accurately classify emails as either spam or non-spam. By developing this detector, we aim to enhance email security and protect users from falling victim to spam emails. .

3. Data Checks to Perform 🌏

3.1 Import Necessary Libraries

In [153…

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)

import plotly.tools as tls
from plotly.tools import FigureFactory as FF
```

```
import warnings
warnings.filterwarnings('ignore')
```

3.2 Load the Data

In [154…
```
df = pd.read_csv(r"C:\Users\Dell\OneDrive\Desktop\PRACTICE\spam detection\archive (9)\
df
```

Out[154]:

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

5572 rows × 5 columns

In [155…
```
styled_df = df.head()
styled_df = styled_df.style.set_table_styles([
    {"selector": "th", "props": [("color", 'black'), ("background-color", '#87CEEB')]}
])

styled_df
```

Out[155]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	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 final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	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 around here though	nan	nan	nan

4. Data Cleaning

4.1 | Data Info

In [156…
```
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
```

4.2 | Drop the Columns

In [157…
```
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'), ("background-color", "#87CEEB"),
])
```

Out[157]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	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 final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	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 around here though	nan	nan	nan

4.3 | Rename the Column

In [158…
```
 # Rename the columns name
df.rename(columns = {'v1': 'target', 'v2': 'text'}, inplace = True)
```

4.4 | Convert the target variable

In [159…
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['target'] = encoder.fit_transform(df['target'])
```

In [160…
```
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'), ("background-color", "#87CEEB"),
])
```

Out[160]:

	target	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	nan	nan	nan
1	0	Ok lar... Joking wif u oni...	nan	nan	nan
2	1	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	nan	nan	nan
3	0	U dun say so early hor... U c already then say...	nan	nan	nan
4	0	Nah I don't think he goes to usf, he lives around here though	nan	nan	nan

4.5 | Check Missing values

In [161…
```
#checking missing values
df.isnull().sum()
```

Out[161]:
```
target           0
text             0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64
```

4.6 | Check Duplicate values

In [162…
```
#check duplicate values
df.duplicated().sum()
```

Out[162]:
```
403
```

4.7 | Remove Duplicate values

In [163…
```
#remove Duplicate
df = df.drop_duplicates(keep = 'first')
```

4.8 | Shape of the Dataset

In [164…
```
df.shape
```

Out[164]:
```
(5169, 5)
```

5. EDA

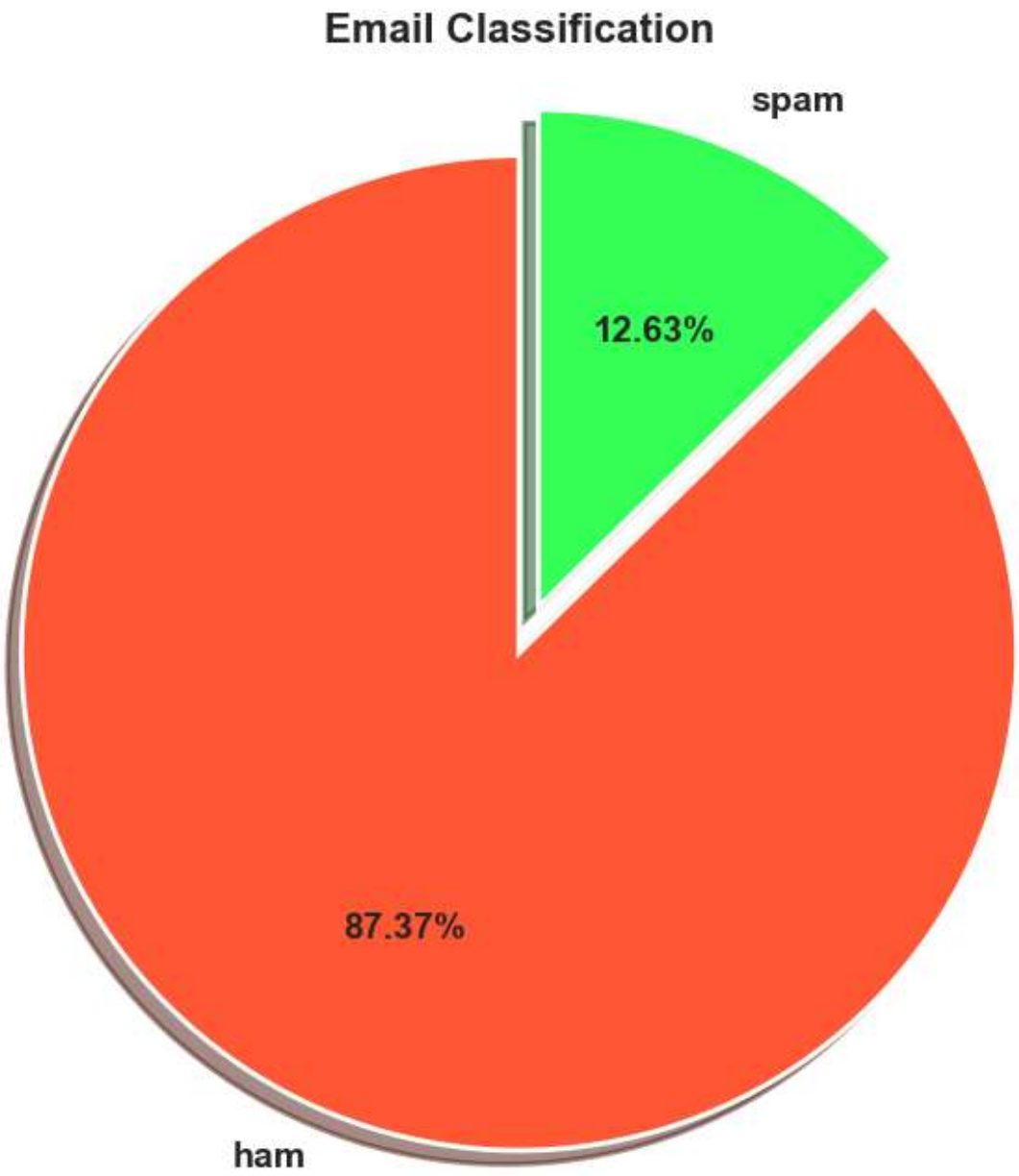
5.1 | Percentage of Ham and Spam

In [165…
```
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 : 87.3669955503966
percentage of 1 : 12.633004449603405
```

In [166…
```
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, 'edgecolor': '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()
```



As you see to the graph the percentage of ham is too high (87.37%) as compare to spam messages percentage. so the data is imbalance

5.2 | Text Length and Structure Analysis

In [167…

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[167]:
```
True
```

In [168…
```
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)))
```

In [169…
```
df[['num_characters', 'num_words', 'num_sentence']].describe()
```

Out[169]:

	num_characters	num_words	num_sentence
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

5.3 | Summary Statistics for Legitimate Messages

In [170…
```
#ham
df[df['target'] == 0][['num_characters', 'num_words', 'num_sentence']].describe()
```

Out[170]:

	num_characters	num_words	num_sentence
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

5.4 | Summary Statistics for Spam Messages

In [171…
```
#spam
df[df['target'] == 1][['num_characters', 'num_words', 'num_sentence']].describe()
```

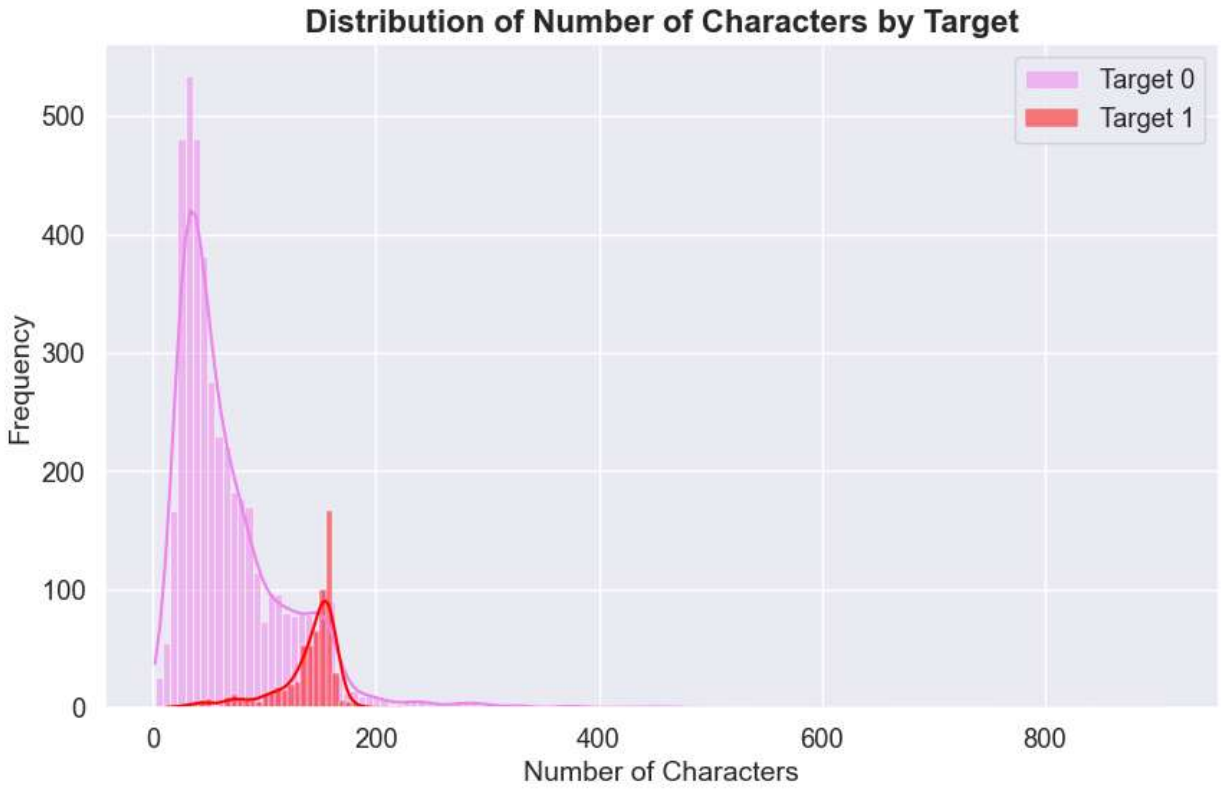
Out[171]:

	num_characters	num_words	num_sentence
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

5.5 | Character Length Distribution for Legitimate and Spam Messages

In [172…

```
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='violet', label='Target 0'

# Plot the histogram for target 1 in red
sns.histplot(df[df['target'] == 1]['num_characters'], color='red', label='Target 1', k

# 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='b

# 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()
```



5.6 | Word Count Distribution for Legitimate and Spam Messages

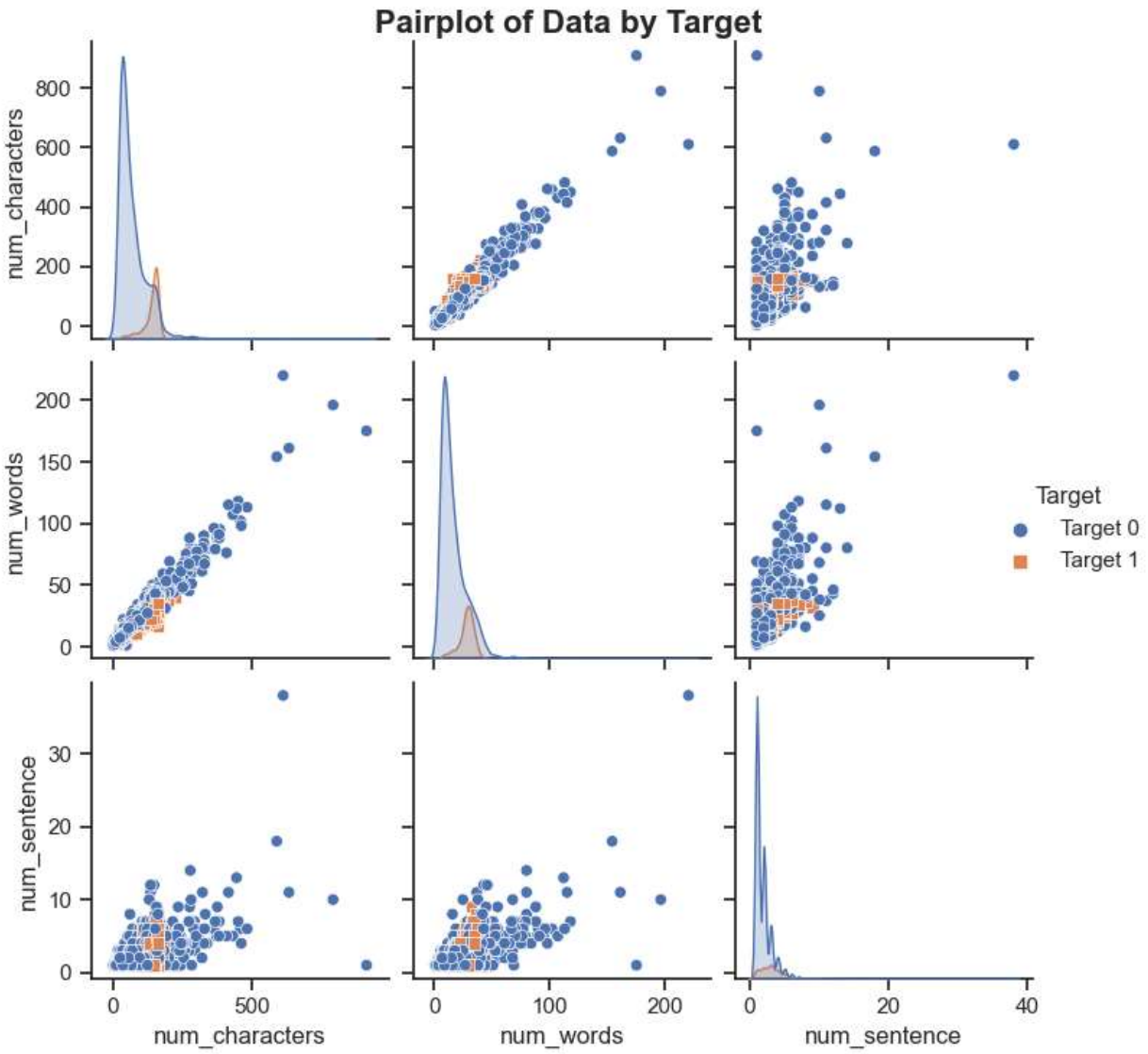
In [173…

```
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=T

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

# 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()
```

5.7 | Pairplot for Data Visualization

In [174…

```
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()
```



5.8 | Coorelation

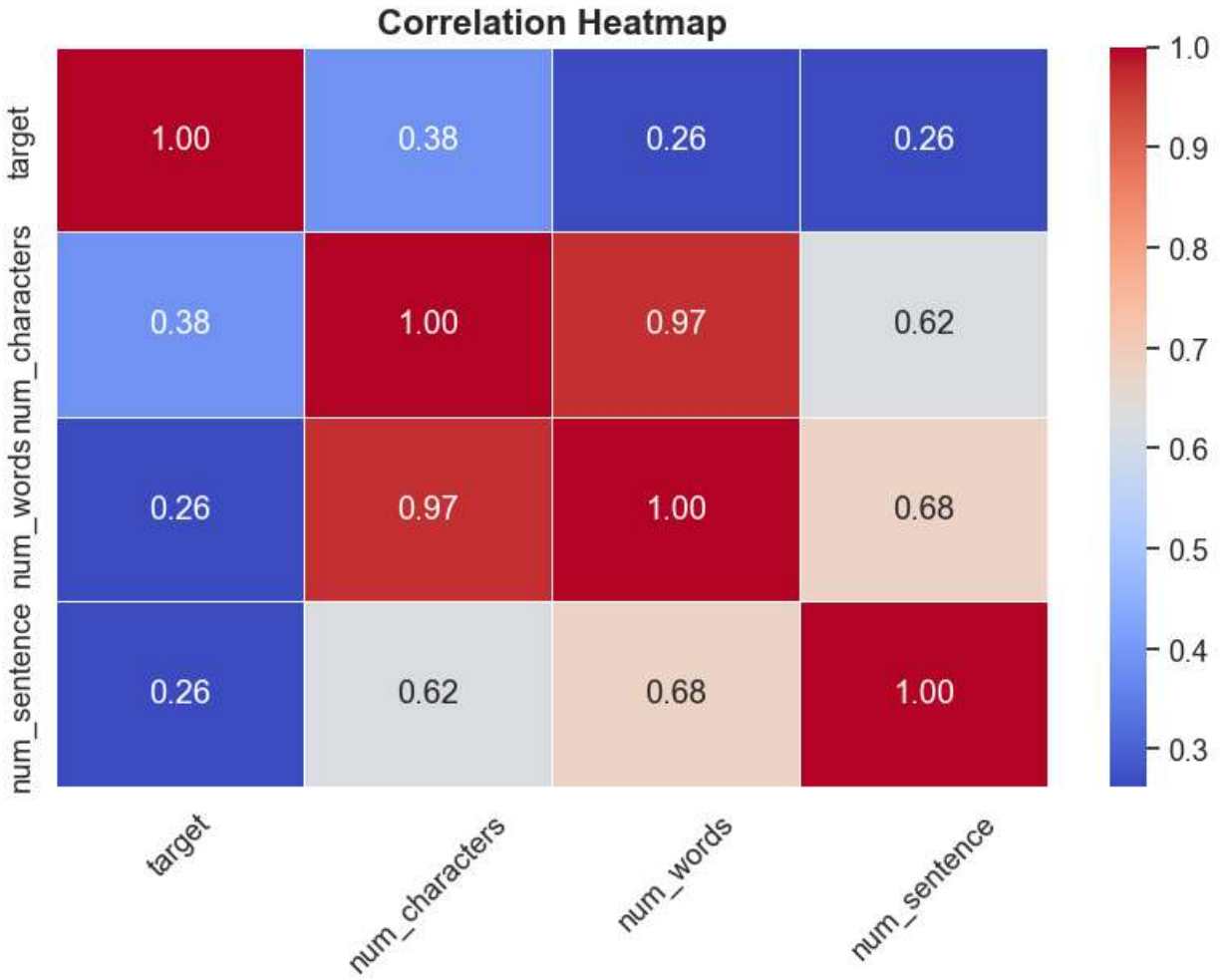
In [175…
```
df[['target','num_characters', 'num_words', 'num_sentence']].corr()
```

Out[175]:

	target	num_characters	num_words	num_sentence
target	1.000000	0.384717	0.262912	0.263939
num_characters	0.384717	1.000000	0.965760	0.624139
num_words	0.262912	0.965760	1.000000	0.679971
num_sentence	0.263939	0.624139	0.679971	1.000000

In [176…
```
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']].cor

# 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)

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



6. Data Preprocessing

In [177…

```
# 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)
```

In [178…

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

# Download NLTK data for stopwords (if not already downloaded)
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[178]:

```
True
```

In [179…

```
transform_text('Go until jurong point, crazy.. Available only in bugis n great world l
```

Out[179]:

```
'go jurong point crazi avail bugi n great world la e buffet cine got amor wat'
```

6.1 | Creating a New Column: 'transformed_text'

In [180…

```
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'), ("background-color", "#87CEEB"),
])
```

Out[180]:

	target	text	Unnamed: 2	Unnamed: 3	Unnamed: 4	num_characters	num_words	num_s
0	0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	nan	nan	nan	111	24	
1	0	Ok lar... Joking wif u oni...	nan	nan	nan	29	8	
2	1	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	nan	nan	nan	155	37	
3	0	U dun say so early hor... U c already then say...	nan	nan	nan	49	13	
4	0	Nah I don't think he goes to usf, he lives around here though	nan	nan	nan	61	15	

6.2 | Word Cloud for Spam Messages

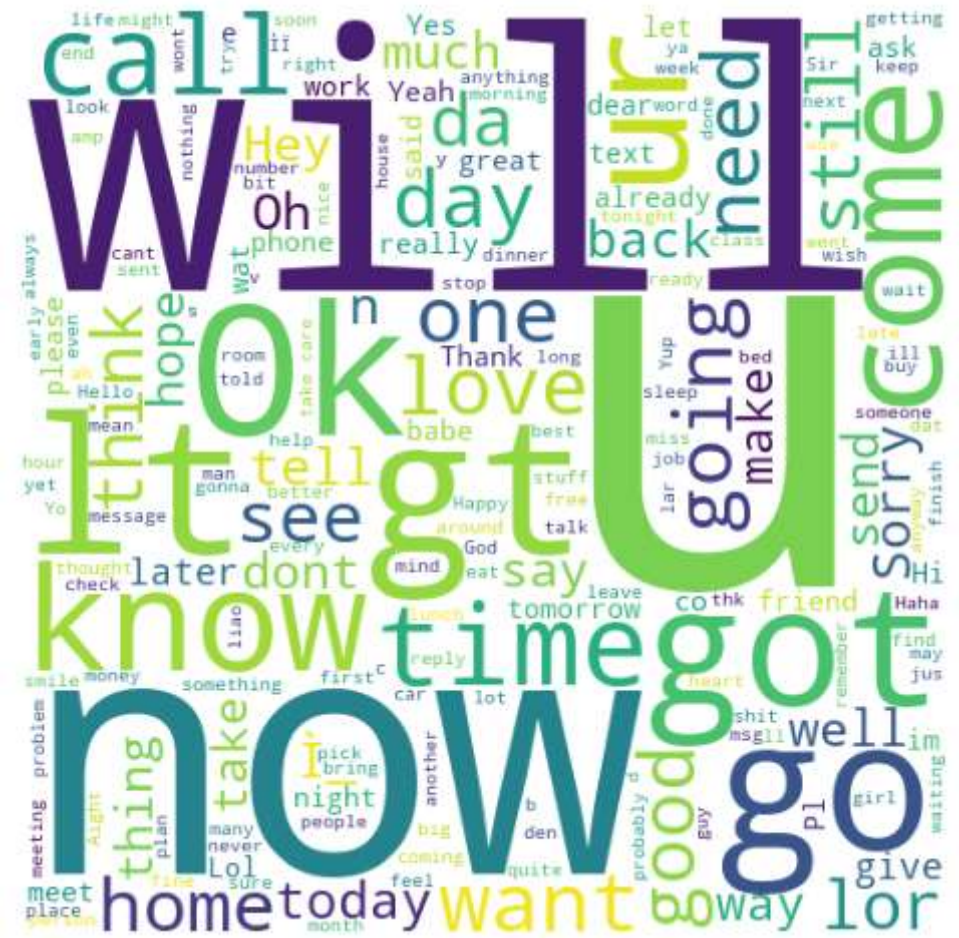
In [181…

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a WordCloud object
wc = WordCloud(width=500, height=500, min_font_size=10, background_color='white')

# Generate a word cloud from the 'text' column of spam messages (where target == 1)
spam_wc = wc.generate(df[df['target'] == 1]['text'].str.cat(sep=" "))

# Plot the word cloud
plt.figure(figsize=(15, 6))
plt.imshow(spam_wc)
plt.axis("off")
plt.show()
```

6.2 | Word Cloud for Not spam Messages

In [182…

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a WordCloud object
wc = WordCloud(width=500, height=500, min_font_size=10, background_color='white')

# Generate a word cloud from the 'text' column of ham messages (where target == 0)
ham_wc = wc.generate(df[df['target'] == 0]['text'].str.cat(sep=" "))

# Plot the word cloud
plt.figure(figsize=(15, 6))
plt.imshow(ham_wc)
plt.axis("off")
plt.show()
```



6.3 | Find top 30 words of spam

In [183…
```
spam_words = []

# Iterate through the 'text' column of spam messages (where target == 1)
for sentence in df[df['target'] == 1]['text'].tolist():
    for word in sentence.split():
        spam_words.append(word)

# Now, you have a list 'spam_words' containing words from spam messages
```

In [184…
```
from collections import Counter
filter_df = pd.DataFrame(Counter(spam_carpos).most_common(30))
```

In [185…
```
# Assuming your DataFrame is named 'filter_df'
column_names = filter_df.columns
print(column_names)
```

```
Index([], dtype='object')
```

6.4 | Find top 30 words of Not spam Messages

In [186…
```
# List the column names in your DataFrame
print(df.columns)
```

```
# Select the correct column containing text data
text_column = 'column_name_containing_text'  # Replace with the actual column name

# Now, you can use this 'text_column' in your code
```

```
Index(['target', 'text', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4',
       'num_characters', 'num_words', 'num_sentence'],
      dtype='object')
```

In [187…
```
ham_carpos = []
for sentence in df[df['target'] == 0]['text'].tolist():
    for word in sentence.split():
        ham_carpos.append(word)
```

In [188…
```
filter_ham_df = pd.DataFrame(Counter(spam_carpos).most_common(30))
```

7. Model Building

7.1 | Initializing CountVectorizer and TfidfVectorizer

In [189…
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
cv = CountVectorizer()
tfid = TfidfVectorizer(max_features = 3000)
```

7.2 | Dependent and Independent Variable

In [190…
```
X = tfid.fit_transform(df['text']).toarray()
y = df['target'].values
```

7.3 | Split into Train and Test Data

In [191…
```
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_stat
```

7.4 | Import the Models

In [192…
```
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
```

7.5 | Initialize the Models

In [193…

```
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)
```

7.6 | Dictionary of the Models

In [194…

```
clfs = {
    'SVC': svc,
    'KNN': knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'Adaboost': abc,
    'Bgc': bc,
    'ETC': etc,
    'GBDT': gbdt,
    'xgb': xgb

}
```

7.7 |Train the Models

In [195…

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

8. Evaluate the Models

In [196…

```
accuracy_scores = []
precision_scores = []
for name , clfs in clfs.items():
    current_accuracy, current_precision = train_classifier(clfs, X_train, y_train, X_t
    print()
    print("For: ", name)
    print("Accuracy: ", current_accuracy)
    print("Precision: ", current_precision)
```

```
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

```
For:  SVC
Accuracy:  0.9806576402321083
Precision:  0.9836065573770492

For:  KNN
Accuracy:  0.90715667311412
Precision:  1.0

For:  NB
Accuracy:  0.9738878143133463
Precision:  1.0

For:  DT
Accuracy:  0.941972920696325
Precision:  0.8979591836734694

For:  LR
Accuracy:  0.960348162475822
Precision:  0.944954128440367

For:  RF
Accuracy:  0.9748549323017408
Precision:  1.0

For:  Adaboost
Accuracy:  0.9748549323017408
Precision:  0.9242424242424242

For:  Bgc
Accuracy:  0.9622823984526112
Precision:  0.896

For:  ETC
Accuracy:  0.9787234042553191
Precision:  0.9833333333333333

For:  GBDT
Accuracy:  0.9516441005802708
Precision:  0.9313725490196079

For:  xgb
Accuracy:  0.9796905222437138
Precision:  0.9834710743801653
```

9. Conclusion

Conclusion: In our evaluation of various classification algorithms, we observed the following key insights:

Support Vector Classifier (SVC) and Random Forest (RF) demonstrated the highest accuracy, both achieving approximately 97.58%. Naive Bayes (NB) achieved a perfect precision score, indicating zero false positives. Other models, including Gradient Boosting, Adaboost, Logistic Regression, and Bagging Classifier, displayed competitive performance with accuracy scores

ranging from 94.68% to 96.03%. The selection of the optimal model should consider factors beyond just accuracy, such as computational efficiency and the specific requirements of the application. It is advisable to perform further model fine-tuning and validation before making a final choice.

10. Author Message

Author : Bishwajit Sen 👉Shoot me mails : sen.bishwajit@rediffmail.com 👉Connect on LinkedIn : https://www.linkedin.com/in/bishwajit-sen/ 👉Explore Github : https://github.com/BISHWAJITSEN If you liked this Notebook, please do upvote. If you have any questions, feel free to comment! ✨Best Wishes✨