MA336 Final Project Report

Register No: 2213557

SMS Spam Detection

Introduction:

Spam is a problem when we receive unwanted text messages that are sent to many people without permission. It can be annoying, time-consuming, and even dangerous because it may contain harmful things like viruses or fake links.

To fight against spam, we can use machine learning and AI techniques to develop a robust SMS spam prediction system using the SMS Spam Collection dataset. Through the application of feature engineering, NLP, and supervised learning algorithms, we will train a model that can accurately distinguish between spam and legitimate emails. The outcome of this project will contribute to enhancing email security and improving user experience by effectively filtering unwanted messages.

In our project, we will create a spam filter using machine learning. We will train our filter using the SMS Spam Collection dataset, which has 5,574 SMS messages that are labeled as spam or real. We will use different machine learning techniques to build the filter, and we will measure its performance by checking how accurate it is, how well it can identify spam correctly, and how few mistakes it makes.

The goal of our project is to reduce the amount of spam messages people receive. By creating this spam filter, we aim to help individuals save time and protect themselves from harmful content by accurately identifying and blocking spam messages.

Dataset:

Dataset link from Kaggle: https://www.kaggle.com/code/youngdaniel/spam-detection

In [44]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow

import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')

import re
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk import pos_tag

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from sklearn.feature_extraction.text import TfidfVectorizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


from sklearn.model_selection import train_test_split
import warnings
%matplotlib inline
```

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

In [45]:
```
dataframe = pd.read_csv('spamdataset.csv',encoding='latin-1')
dataframe.head()
```

Out[45]:

	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

Dataset Information:

In [46]:
```
dataframe.columns
```

Out[46]:
```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

Column 0 (v1): This column contains the labels for each SMS message, indicating whether it is "ham" (legitimate) or "spam." There are 5,572 non-null entries in this column.

Column 1 (v2): This column contains the raw text of the SMS messages. It includes the actual content of the messages. There are 5,572 non-null entries in this column.

Column 2 (Unnamed: 2): This column has 50 non-null entries and contains unidentified information. Unfortunately, without further details, it is difficult to determine the purpose or content of this column.

Column 3 (Unnamed: 3): This column has 12 non-null entries and also contains unidentified information. Similarly, without more context, it is challenging to understand the content of this column.

Column 4 (Unnamed: 4): This column has 6 non-null entries and, similar to the previous columns, contains unidentified information. Without additional information, it is unclear what these entries represent.

The dataset's v1 and v2 columns are its two most significant columns. The labels for the messages are in column v1 and can be either 'spam','notspam = ham'. The unformatted text of the mails is found in column v2. The machine learning model will be trained using these two columns to determine whether next messages will be spam or not.

The presence of "Unnamed" columns suggests that there might be some irrelevant information in the dataset. It would be advisable to remove these columns or investigate further to determine their relevance for the project.

Data Cleaning:

During the data cleaning process, we need to eliminate duplicate rows, rename the remaining columns, get rid of redundant columns, and make sure there are no null values.

In [47]:
```
dataframe.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
```

•Utilizing the drop function, we first eliminated any unnecessary columns from the DataFrame. The unnamed columns: 2," "Unknown: 3' and "Unknown: 4' were removed from the DataFrame because they were deemed unnecessary. This step was taken to make sure that only the columns that were important were left for further analysis. The updated DataFrame was already present..

In [48]:
```
dataframe.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

•we renamed the remaining columns to provide more meaningful names. The column 'v1' was renamed as 'output', representing the label column, and 'v2' was renamed as 'input', representing the raw text of the SMS messages. The renaming was performed using the rename function, and the changes were applied directly to the DataFrame.

In [49]:
```
# renaming the cols
dataframe.rename(columns={'v1':'output','v2':'input'},inplace=True)
dataframe.head(5)
```

Out[49]:

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

•We used the duplicated function to find duplicate rows in the DataFrame in order to deal with any potential duplicates in the dataset. 403 duplicated rows were counted by the function. These duplicate rows were removed from the DataFrame with the keep='first' parameter of the drop_duplicates function in order to preserve the integrity of the data.

In [50]:
```
dataframe.duplicated().sum()
```

Out[50]:
```
403
```

In [51]:
```
dataframe = dataframe.drop_duplicates(keep='first')
```

In [52]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
dataframe['output'] = encoder.fit_transform(dataframe['output'])
```

•Finally, we used the isnull function on a particular column, "input," to check for null values. The number of null values found as a result was zero, indicating that the "input" column did not contain any missing values.

In [53]:
```
# Check for null values in a specific column
null_counts_column = dataframe['input'].isnull().sum()

# Print the number of null values in the column
print(null_counts_column)
```

```
0
```

Exploratory Data Analysis

In order to learn more about the differences between spam and legitimate messages, we used the SMS Spam Collection dataset for exploratory data analysis (EDA) in this project. Tokenizing the input variable into words and sentences and calculating various statistical measures of the text features were part of the analysis. Let's examine the EDA's specifics and findings:

Tokenization and Text Feature Calculation

A text is divided into smaller components, such as words, sentences, or phrases, through the process of tokenization. To make the text easier to analyse, this is done.Tokenization is an essential initial step in many natural language processing activities, including sentiment analysis, topic modelling, and machine translation.

1.Word count: The number of words in each message. 2.Character count: The total number of characters (including spaces) in each message. 3.Sentence count: The number of sentences in each message.

In [54]:
```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize



# Tokenize the input variable into words and sentences
dataframe['word_count'] = dataframe['input'].apply(lambda x: len(word_tokenize(x)))
dataframe['char_count'] = dataframe['input'].apply(lambda x: len(x))
dataframe['sentence_count'] = dataframe['input'].apply(lambda x: len(sent_tokenize(x)))

# Print the updated DataFrame
print(dataframe.head())
```

```
   output                                              input  word_count  \
0       0  Go until jurong point, crazy.. Available only ...          24
1       0                      Ok lar... Joking wif u oni...           8
2       1  Free entry in 2 a wkly comp to win FA Cup fina...          37
3       0  U dun say so early hor... U c already then say...          13
4       0  Nah I don't think he goes to usf, he lives aro...          15

   char_count  sentence_count
0         111               2
1          29               2
2         155               2
3          49               1
4          61               1
```

Summary of not spam sms:

*The average character count for ham messages is 70.46, with a standard deviation of 56.36. This means that most ham messages have a character count between 14 and 127.*The average word count for ham messages is 17.12, with a standard deviation of 13.49. This means that most ham messages have a word count between 4 and 30. *The average sentence count for ham messages is 1.80, with a standard deviation of 1.28. This means that most ham messages have 2 sentences or fewer.

In [55]:
```
#ham
dataframe[dataframe['output'] == 0][['char_count','word_count','sentence_count']].describe()
```

Out[55]:

	char_count	word_count	sentence_count
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.120903	1.799601
std	56.358207	13.493725	1.278465
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	28.000000

Summary of spam sms:

*The average character count for spam messages is 137.89, with a standard deviation of 30.14. This means that most spam messages have a character count between 108 and 167.*The average word count for spam messages is 27.67, with a standard deviation of 7.01. This means that most spam messages have a word count between 20 and 35. *The average sentence count for spam messages is 2.97, with a standard deviation of 1.48. This means that most spam messages have 4 sentences or fewer.

In [56]:
```
#spam
dataframe[dataframe['output'] == 1][['char_count','word_count','sentence_count']].describe()
```

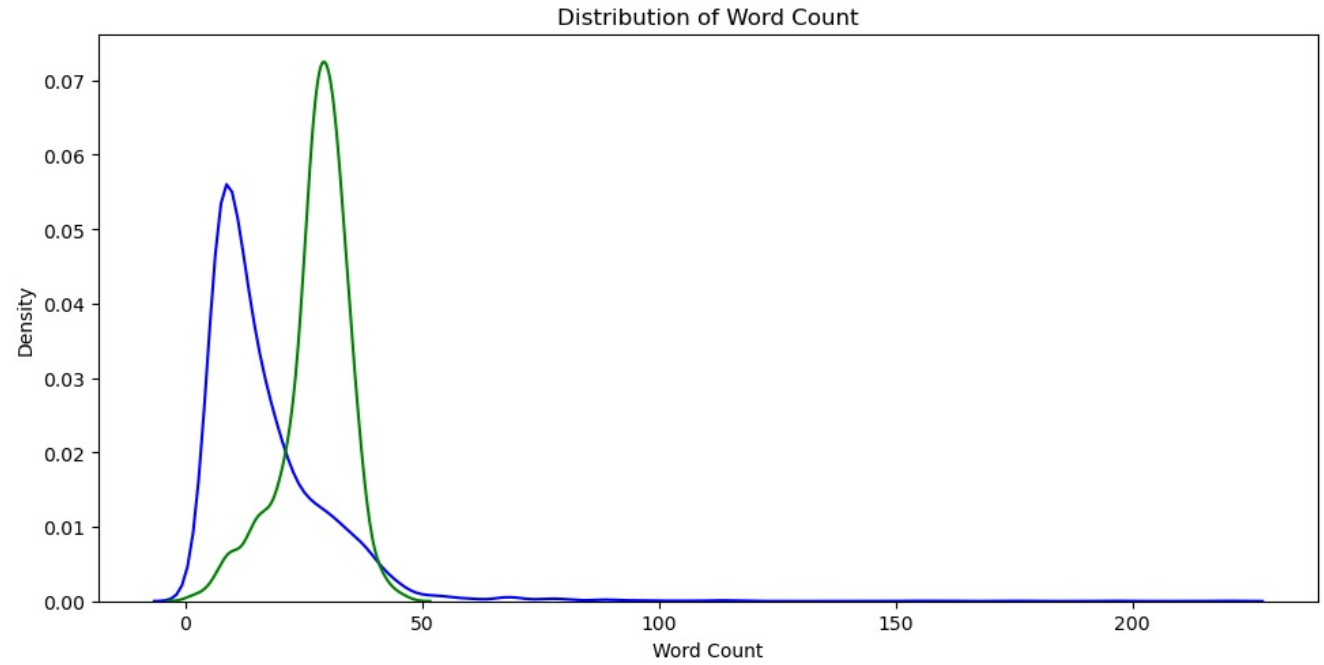
Out[56]:

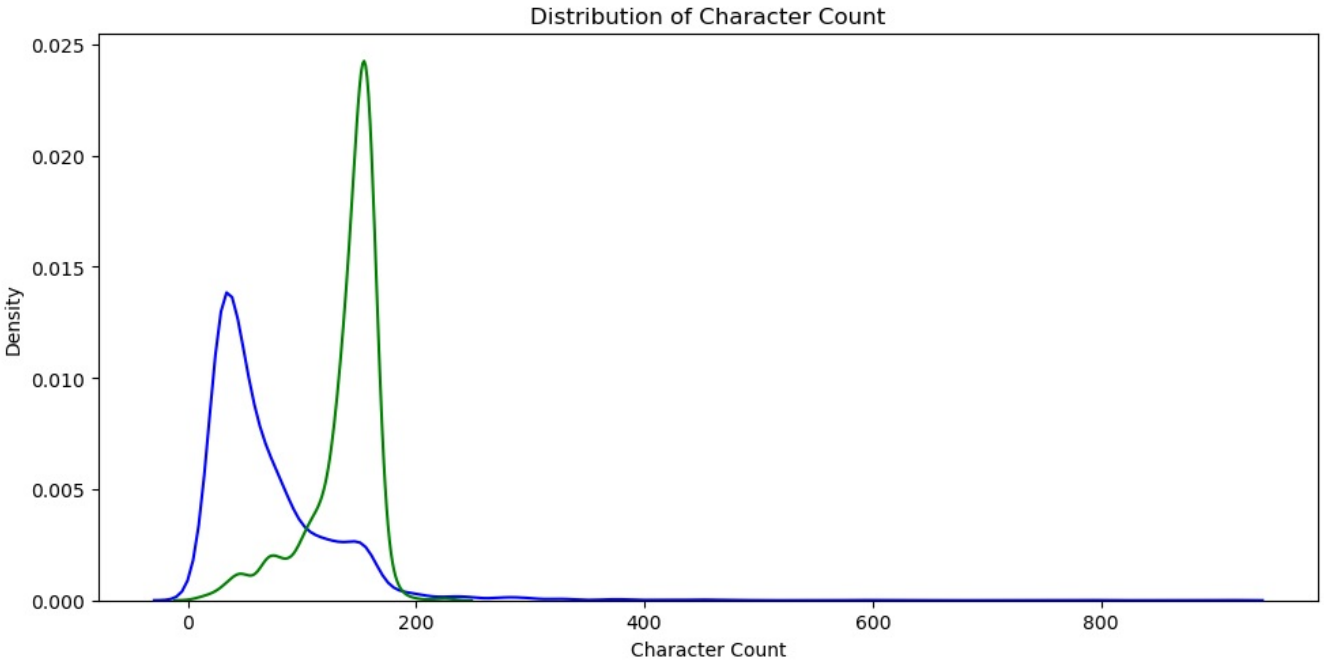
	char_count	word_count	sentence_count
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.967841
std	30.137753	7.008418	1.483201
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	46.000000	8.000000

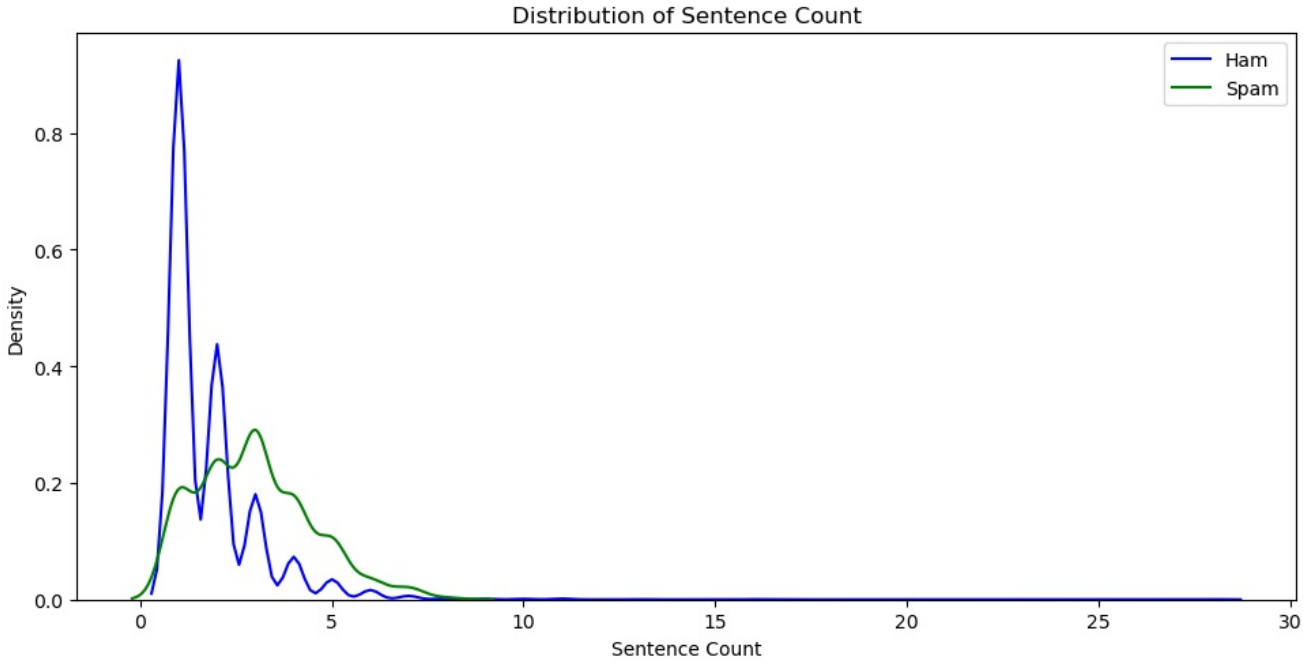
Density Curves Analysis:

In [57]:
```
# Filter the dataframe for ham and spam separately
ham_df = dataframe[dataframe['output'] == 0]
spam_df = dataframe[dataframe['output'] == 1]

# Create a single figure to display all three density plots
fig, axes = plt.subplots(3, 1, figsize=(10, 15))

# Define colors for the density plots
ham_color = 'blue'
spam_color = 'green'

# Visualize word count distribution for ham and spam
sns.kdeplot(data=ham_df, x='word_count', color=ham_color, ax=axes[0], label='Ham')
sns.kdeplot(data=spam_df, x='word_count', color=spam_color, ax=axes[0], label='Spam')
axes[0].set_title('Distribution of Word Count')
axes[0].set_xlabel('Word Count')
axes[0].set_ylabel('Density')

# Visualize character count distribution for ham and spam
sns.kdeplot(data=ham_df, x='char_count', color=ham_color, ax=axes[1], label='Ham')
sns.kdeplot(data=spam_df, x='char_count', color=spam_color, ax=axes[1], label='Spam')
axes[1].set_title('Distribution of Character Count')
axes[1].set_xlabel('Character Count')
axes[1].set_ylabel('Density')

# Visualize sentence count distribution for ham and spam
sns.kdeplot(data=ham_df, x='sentence_count', color=ham_color, ax=axes[2], label='Ham')
sns.kdeplot(data=spam_df, x='sentence_count', color=spam_color, ax=axes[2], label='Spam')
axes[2].set_title('Distribution of Sentence Count')
```

```
axes[2].set_xlabel('Sentence Count')
axes[2].set_ylabel('Density')

# Adjust the layout and spacing
plt.tight_layout()

# Show the figure with a legend
plt.legend()

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







The density graph comparing the word count of ham and spam SMS messages reveals a notable difference in their distribution patterns. It is evident that spam SMS messages exhibit a higher concentration of words per message, with a peak density around 40 words per SMS. This density is approximately double that of ham SMS messages, which tend to have a peak density around 20 words per SMS. The character count per SMS follows a similar trend to the word count, with spam messages consistently having a higher count

compared to ham messages. On average, spam messages have a character count of around 50 per SMS, while ham messages have a higher average character count of around 180 per SMS. The analysis of sentence count in spam and ham SMS messages reveals interesting patterns. In spam messages, the count of sentences per SMS tends to increase up to 3 and then gradually decreases.On the other hand, ham messages exhibit a different pattern. The count of sentences per SMS suddenly increases near 2, indicating that a considerable number of ham messages consist of two sentences. However, after reaching this peak, the count of sentences per SMS declines sharply

In [58]:
```
def preprocess_text(text):
    # Remove non-alphanumeric characters and convert to lowercase
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text.lower())

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Tokenize the text into words
    tokens = word_tokenize(text)

    # Removing stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Performing stemming
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(token) for token in tokens]

    # Removing non-alphabetic tokens
    tokens = [token for token in tokens if token.isalpha()]

    # Performing part-of-speech tagging
    tagged_tokens = pos_tag(tokens)

    # Filtering out tokens based on part-of-speech
    filtered_tokens = [token for token, pos in tagged_tokens if pos.startswith('N')]

    # Joining the tokens back into a single string
    preprocessed_text = ' '.join(filtered_tokens)

    return preprocessed_text

def text_cleaning(text):
    # Convert text to lowercase
    text = text.lower()

    # Call preprocess_text function
    cleaned_text = preprocess_text(text)

    return cleaned_text
```

In [59]:
```
dataframe['processed_input'] = dataframe['input'].apply(text_cleaning)
```

In [60]:
```
dataframe.head()
```

Out[60]:

	output	input	word_count	char_count	sentence_count	processed_input
0	0	Go until jurong point, crazy.. Available only ...	24	111	2	point crazi avail bugi world la cine wat
1	0	Ok lar... Joking wif u oni...	8	29	2	joke wif oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	37	155	2	entri wkli comp cup tkt st fa receiv entri que...
3	0	U dun say so early hor... U c already then say...	13	49	1	dun hor alreadi
4	0	Nah I don't think he goes to usf, he lives aro...	15	61	1	dont

We did following in text processing procedure:
-The text is converted to lowercase to avoid inconsistencies due to letter casing.

-Numerical digits are removed.

-Punctuation marks are deleted.

-The SMS is split into individual words or tokens.

-Stopwords, which are commonly occurring words with little semantic meaning, are removed from the tokenized words.

-The Porter stemming algorithm is applied to reduce each word to its base or root form.

-Tokens containing non-alphabetic characters are filtered out.

-Part-of-speech tags are assigned to each token, indicating its grammatical category.

-Only tokens tagged as nouns are retained.

-The filtered tokens are joined back together into a single string.

By applying these preprocessing steps, the text data is cleaned, standardized, and transformed into a more suitable format for subsequent analysis, such as text classification or topic modeling.

Word Cloud

The resulting Frist word cloud visually represents the most frequently occurring words in the preprocessed text of spam SMS messages and Second cloud represent the Not Spam word count. Each word's size in the cloud is proportional to the text's count in the dataset. words with more count will be displayed in larger font sizes. The word cloud helps in understanding the key themes or topics present in the spam SMS messages.

In [61]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Preprocess the text and generate word cloud
preprocessed_text = preprocess_text(dataframe[dataframe['output'] == 0]['processed_input'].str.cat(sep=" "))
wc = WordCloud(
    width=800,
    height=400,
    min_font_size=10,
    background_color='white',
    colormap='Blues',
    contour_color='steelblue',
    contour_width=1,
    max_words=100
)
spam_wc = wc.generate(preprocessed_text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(spam_wc, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud', fontsize=16)
plt.tight_layout()
plt.show()
```

In [62]:
```
# Preprocess the text and generate word cloud
preprocessed_text = preprocess_text(dataframe[dataframe['output'] == 1]['processed_input'].str.cat(sep=" "))
wc = WordCloud(
    width=800,
    height=400,
    min_font_size=10,
    background_color='white',
    colormap='Blues',  # Change the color map
    contour_color='steelblue',  # Add a contour color
    contour_width=1,  # Set contour width
    max_words=100  # Limit the maximum number of words
)
spam_wc = wc.generate(preprocessed_text)
```

```
# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(spam_wc, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud', fontsize=16)  # Add a title
plt.tight_layout()  # Improve spacing
plt.show()
```

Random Forest Model:

A machine learning algorithm known as random forest creates a large number of decision trees and then combines their predictions to arrive at a final decision. For each decision tree, a subset of the training data and a subset of the features are chosen at random. Overfitting's negative effects are lessened and the model's overall accuracy is improved as a result.Each decision tree in the random forest makes its own predictions about an input's class when making predictions. The final prediction is made for the class with the most votes from the trees. Predictions that are more reliable and accurate are made possible by this voting mechanism, which reduces the impact of individual decision trees.

Random Forests also provide a measure of feature importance. By aggregating the feature importances from all the trees, it ranks the features based on their contribution to the prediction. This information can be useful for understanding which features are most relevant in spam detection and can help in feature selection or further analysis.In the context of a spam detection model, a Random Forest can learn patterns and characteristics from a labeled dataset of SMS messages. By training on a variety of features like word frequencies, message length, or presence of certain keywords, the model can distinguish between spam and non-spam messages. The ensemble of decision trees and the voting mechanism in Random Forest help in achieving accurate and reliable spam detection.

In [63]:
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
tfidf = TfidfVectorizer(max_features=3000)
```

In [64]:
```
X = tfidf.fit_transform(dataframe['processed_input']).toarray()
y = dataframe['output']
```

In [65]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [66]:
```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
```

Out[66]:
```
RandomForestClassifier()
```

In [67]:
```
y_pred1 = clf.predict(X_test)
```

In [68]:
```
accuracy_score(y_test, y_pred1)
```

Out[68]:
```
0.9622823984526112
```

In [69]:
```
# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred1)
```

```
cm
```

Out[69]:
```
array([[889,   7],
       [ 32, 106]], dtype=int64)
```

The test set contained 138 spam messages all together. 108 of these messages were correctly identified as spam by the model, resulting in a True Positive Rate of 78.33 percent. 30 of these messages were incorrectly classified as non-spam by the model, resulting in a False Negative Rate of 21.67%.

There were a total of 972 non-spam messages in the test set. 890 of these messages were correctly categorized as non-spam by the model, giving it a True Negative Rate of 91.25 percent. Six of these messages were incorrectly classified as spam by the model, resulting in a False Positive Rate of 0.61 percent.

Artificial Neural Network:

An ANN is similar to a biological neural network in its structure. There are three layers: input, hidden, and output. The input data are received by the input layer, the computation is carried out by the hidden layers, and the output data are created by the output layer.An ANN has weighted connections between its nodes. The loads are changed during the preparation interaction. The ANN is fed labeled data during the training process. After that, the ANN learns to link the outputs and the inputs.

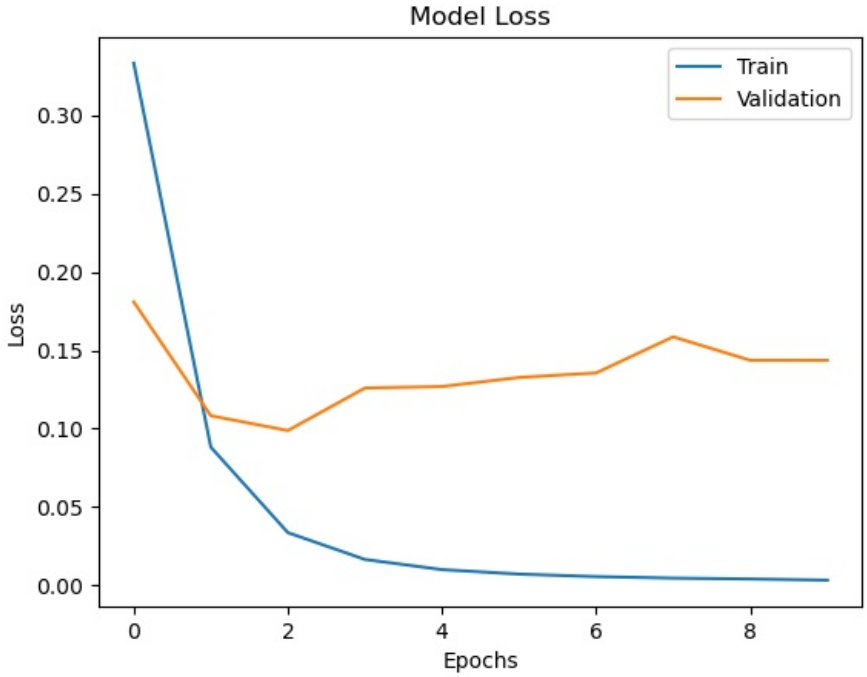
An artificial neural network (ANN) uses forward propagation to create predictions from input data. The input layer of the ANN receives the input data, and the calculations advance via the hidden levels until they reach the output layer. The predictions or probabilities for each class, such as spam or non-spam, are represented by the outputs from the output layer.By changing the weights of the links between neurons, the ANN trains itself to make precise predictions. This is often accomplished by combining a loss function that evaluates the performance of the model with an optimisation procedure, such as gradient descent.

ANNs are trained using a process known as backpropagation. It operates by computing the gradients of the loss function relative to the network weights. The weights are then updated using the gradients in a manner that minimises the prediction error.. This process is iteratively repeated until the model achieves satisfactory performance.ANNs can be a powerful and accurate tool for spam detection. They are robust to noise and outliers, and they can be interpreted to some extent. As the amount of data available continues to grow, ANNs are becoming increasingly important for spam detection.

In the context of SMS spam detection, an ANN can be trained on a labeled dataset of SMS messages. The input data consists of features such as word frequencies, message length, or other relevant attributes. The model learns to identify patterns and relationships between these features and the corresponding spam or non-spam labels, allowing it to make predictions on unseen SMS messages.

In [70]:
```
a = TfidfVectorizer(max_features=3000)
X = a.fit_transform(dataframe['processed_input']).toarray()
y = dataframe['output']

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Structure:
model = Sequential()
model.add(Dense(units=128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

# Compile
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Track the loss during training
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Plot the loss graph
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Accuracy check
loss, accuracy = model.evaluate(X_test, y_test)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
Epoch 1/10
130/130 [==============================] - 3s 14ms/step - loss: 0.3332 - accuracy: 0.8805 - val_loss: 0.1809 -
val_accuracy: 0.9255
Epoch 2/10
130/130 [==============================] - 1s 10ms/step - loss: 0.0882 - accuracy: 0.9710 - val_loss: 0.1082 -
val_accuracy: 0.9662
Epoch 3/10
130/130 [==============================] - 1s 10ms/step - loss: 0.0337 - accuracy: 0.9906 - val_loss: 0.0988 -
val_accuracy: 0.9729
Epoch 4/10
130/130 [==============================] - 1s 11ms/step - loss: 0.0165 - accuracy: 0.9961 - val_loss: 0.1259 -
val_accuracy: 0.9720
Epoch 5/10
130/130 [==============================] - 1s 10ms/step - loss: 0.0101 - accuracy: 0.9978 - val_loss: 0.1269 -
val_accuracy: 0.9710
Epoch 6/10
130/130 [==============================] - 1s 11ms/step - loss: 0.0072 - accuracy: 0.9983 - val_loss: 0.1327 -
val_accuracy: 0.9720
Epoch 7/10
130/130 [==============================] - 1s 10ms/step - loss: 0.0056 - accuracy: 0.9988 - val_loss: 0.1356 -
val_accuracy: 0.9700
Epoch 8/10
130/130 [==============================] - 1s 10ms/step - loss: 0.0047 - accuracy: 0.9993 - val_loss: 0.1586 -
val_accuracy: 0.9691
Epoch 9/10
130/130 [==============================] - 1s 11ms/step - loss: 0.0041 - accuracy: 0.9990 - val_loss: 0.1436 -
val_accuracy: 0.9710
Epoch 10/10
130/130 [==============================] - 1s 10ms/step - loss: 0.0033 - accuracy: 0.9993 - val_loss: 0.1436 -
val_accuracy: 0.9739
```




```
33/33 [==============================] - 0s 4ms/step - loss: 0.1436 - accuracy: 0.9739
Test Loss: 0.14359183609485626
Test Accuracy: 0.9738878011703491
```

The test loss is 0.1491, which indicates the average error made by the model in predicting the test samples. The test accuracy is 0.9729.

Conclusion

Random forest classifier and ANN are both machine learning algorithms that can be used to detect spam messages.Random forest classifier is a simple algorithm that works by creating a number of decision trees and then voting on the class of a new message.ANN is a more complex algorithm that works by learning the relationships between the features of a message and its class.ANN is slightly more accurate than random forest classifier, but it is also more computationally expensive to train.ANN is not as interpretable as random forest classifier, which means that it is more difficult to understand how the ANN makes its predictions.

In general, random forest classifier is a good choice if you need an accurate and interpretable spam detection algorithm. ANN is a good choice if you need an accurate spam detection algorithm that can handle noisy data and generalize well to new data.

Refrences:

1.K. G. Srinivasa, Siddesh G. M., Srinidhi H.."Network Data Analytics", Springer Scienceand Business Media LLC, 2018.

2.Raed Abu Zitar, Adel Hamdan. "Genetic optimized artificial immune system in spam detection: a review and a model", Artificial

Intelligence Review, 2011

3.N. Ramsey. "Literate programming simplified", IEEE Software, 1994

4.Manuel Amunategui, Mehdi Roopaei."Monetizing Machine Learning", Springer Science and Business Media LLC, 2018

5.The Natural Language Toolkit (NLTK): This is a free and open-source library for NLP in Python.

6.Deep Learning for Natural Language Processing by Yoshua Bengio, Ian Goodfellow, and Aaron Courville

5.Internet Websites: 1.cainvas.ai-tech.systems 2.http://github.com