Name: Aruna Balasiva
Email: aruna.316@gmail.com
Course: Specialist Certificate in Data Analytics Essentials
GitHub Link: https://github.com/ArunaAR/UCDPA_ArunaBalasiva

Abstract

SMS messages can be classified as either SPAM or HAM.SPAM messages are unsolicited or unknown texts, typically sent to users' or customers' mobile phones for commercial purposes such as promotions or advertising. Unfortunately, scammers often exploit these messages to trick users into revealing personal details like bank information, social security numbers, or home addresses.
HAM messages, on the other hand, are legitimate and known to the user or customer. For example, a user might receive HAM messages if they subscribe to weather updates or billing information; these are considered valid communications.

In this assignment, I will conduct an analysis of a dataset comprising both SPAM and HAM messages, which has been obtained from Kaggle.

Data Import and Pre-processing

For this assignment, PyCharm was used for coding and testing, while Jupyter Notebook was utilized to display dataset outputs such as tables, graphs, and visualizations. The Scikit-learn library was applied for machine learning tasks. The initial step involved importing the dataset, which consisted of a single CSV file named "spam.csv," sourced from https://www.kaggle.com/code/karanchinchpure/spam-sms-email-classification-98-accuracy/data?select=spam.csv.

```
In [2]: #Data Import, Preprocessing
        spamham_data = pd.read_csv(r"C:\Users\aruna\OneDrive\Desktop\UCDPA_Assignment\spam.csv", encoding="ISO-8859-1")
```

I set encoding="ISO-8859-1" due to this error:

```
UnicodeDecodeError: 'utf-8' codec can't decode bytes in position 606-607: invalid continuation byte
```

Upon review, the dataset was found to contain special characters including 'Õ', 'å', and 'Û'.
Additionally, I verified that the dataset comprises 5,572 rows and 5 columns.

```
# number of rows and columns in this dataset
print (spamham_data.shape)

(5572, 5)
```

Next, the .info() method is used to review the data types present in the dataset and check for any null values (NaN) in the columns. The output shows the specific data types for each column and indicates that there are null values in the dataset.

```
# print columns names and dataType.There's no null value in this dataset
spamham_data.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
```

Columns Unnamed: 2, Unnamed: 3, and Unnamed: 4 contain only NAN values.

Out[3]:

	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

As these three columns are not subject to analysis, I have chosen to eliminate them from the dataset.

```
# removed Column that start with U
spamham_data=spamham_data.loc[:,~spamham_data.columns.str.contains('^U')]
```

I have updated the table labels from V1 and V2 to "categories" and "messages".

```
#Rename Column V1 and V2
spamham_data = spamham_data.rename(columns={spamham_data.columns[0]: 'catergories', spamham_data.columns[1]: 'messages'})
```

```
spamham_data.head(10)
```

	catergories	messages
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...

To ensure there are no null values following the column renaming:

```
#Checking if Null values presents in the table
spamham_data.isnull().sum()
```

```
catergories    0
messages       0
dtype: int64
```

Using value_counts(), I found there are 4825 HAM messages and 747 SPAM messages in the dataset.

```
#Total SPAM and HAM
spamham_data['catergories'].value_counts()
```

```
ham     4825
spam     747
Name: catergories, dtype: int64
```

Approximately 13% of the dataset is SPAM, while 87% consists of non-SPAM emails.

```
#Checking Ratio of catergories HAM and SAPM
print("Not SPAM email Ratio catergory-HAM :",round(len(spamham_data[spamham_data['catergories']=='ham'])/len(spamham_data['caterg
print("Spam Email Ratio catergory-SPAM:",round(len(spamham_data[spamham_data['catergories']=='spam'])/len(spamham_data['catergori
```

```
Not SPAM email Ratio catergory-HAM : 87.0 %
Spam Email Ratio catergory-SPAM: 13.0 %
```

For further analysis, a column titled ‘Length’ was created to measure the size of each message:

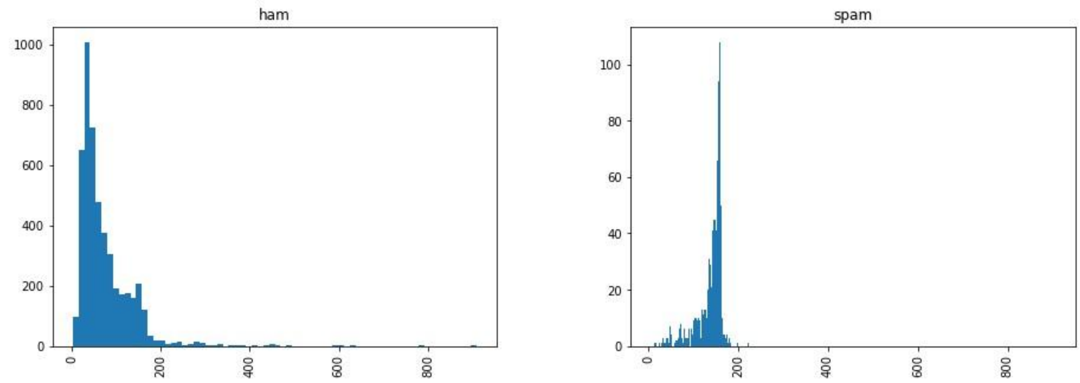
```
#Adding new column Length to the table
spamham_data['Length']=0
for x in np.arange(0,len(spamham_data.messages)):
    spamham_data.loc[x,'Length'] = len(spamham_data.loc[x,'messages'])

spamham_data.head(5)
```

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

Data Visualization:

Next, I use a Plotly Histogram to summarise the dataset by length.

```
#Histogram
spamham_data.hist(column='Length',by='catergories',bins=70,figsize=(15,5));
plt.xlim(-40,950);
```



The histogram indicates that SPAM messages generally exhibit shorter lengths compared to HAM messages.

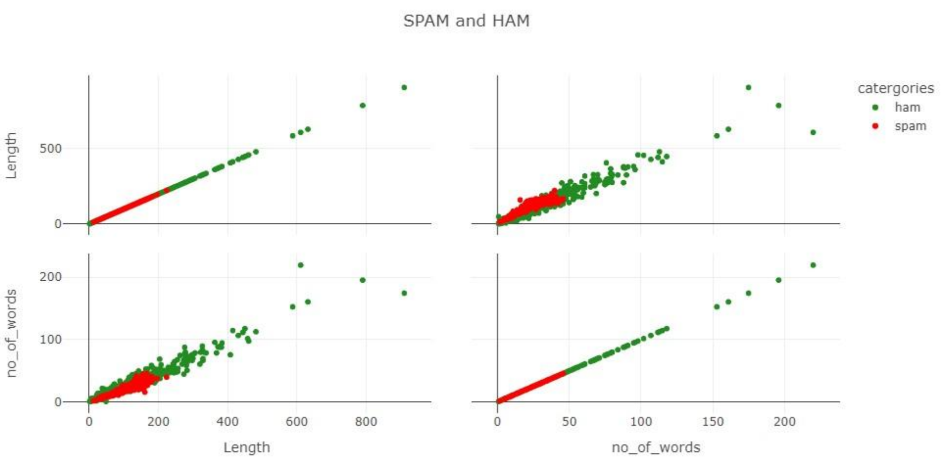
Since the dataset did not include word counts, I added a new column, "no_of_words," for each message.
In Python, strings are case sensitive. For instance:
HELLO == hello is not a true statement.
For simpler analysis, it is common to convert all words to lowercase.

```
#Adding new column no_of_words to the table and converting all the text to lowercase
spamham_data['no_of_words'] = spamham_data['messages'].apply(lambda x: len(nltk.word_tokenize(x)))
spamham_data['messages'] = spamham_data['messages'].apply(lambda x:x.lower())
spamham_data.head(15)
```

	catergories	messages	Length	no_of_words
0	ham	go until jurong point, crazy.. available only ...	111	24
1	ham	ok lar... joking wif u oni...	29	8
2	spam	free entry in 2 a wkly comp to win fa cup fina...	155	37
3	ham	u dun say so early hor... u c already then say...	49	13
4	ham	nah i don't think he goes to usf, he lives aro...	61	15
5	spam	freemsg hey there darling it's been 3 week's n...	148	39
6	ham	even my brother is not like to speak with me. ...	77	18
7	ham	as per your request 'melle melle (oru minnamin...	160	31

I used a scatter plot to quickly review message length and word count in the dataset.

```
In [20]: #Scatter Graph based on Length and No of Words
         fig = px.scatter_matrix(spamham_data, dimensions=["Length",'no_of_words'],
                                 color = "catergories",template='gridon',
                                 color_discrete_map = {'ham': 'forestgreen', 'spam': 'red'},
                                 title = "SPAM and HAM ")
         #fig.update_traces(diagonal_visible=False)
         fig.show()
```

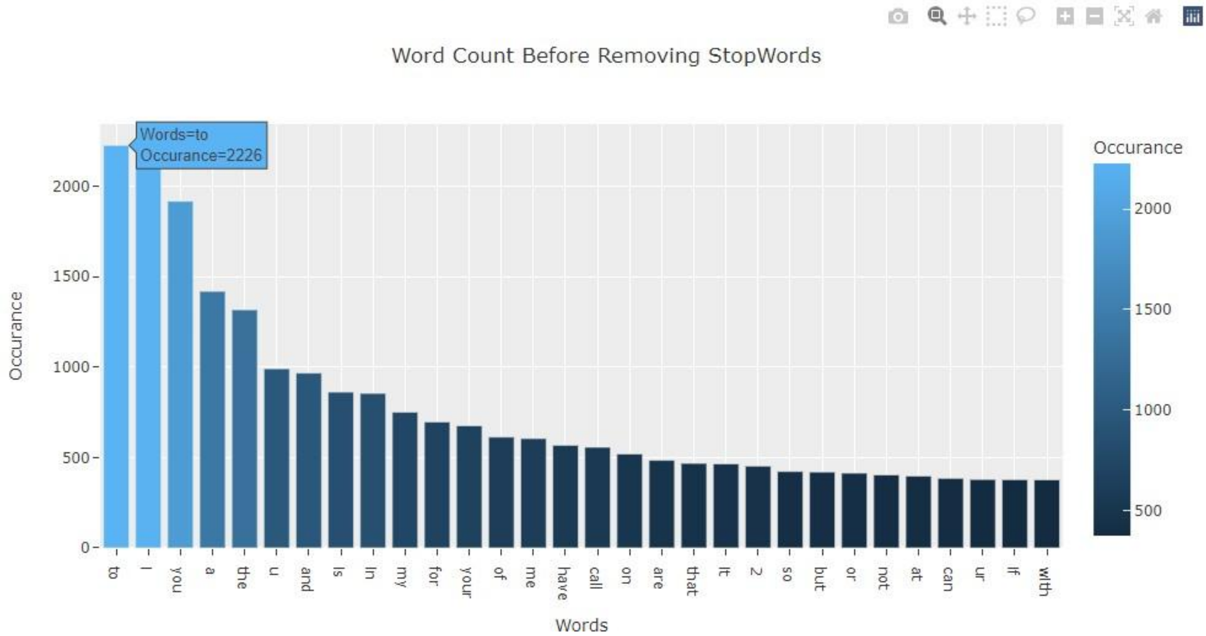


SPAM messages usually contain fewer words than HAM messages. If we examine the scatter plot of word counts for SPAM versus HAM messages, we notice that most SPAM messages cluster toward the lower end of the word count axis, while HAM messages are more spread out and tend to have higher word counts. The separation between these two groups is visually apparent, with only a few overlap points. This trend suggests that message length can serve as a useful indicator when distinguishing between SPAM and HAM in our dataset. Let’s investigate futher.

In everyday communication, individuals often use stopwords such as “I”, “me”, “my”, “myself”, “we”, “our”, and “ours”.

```
#Word counts before removing stopwords
def word_count_plot(spamham, title):
    # finding words along with count
    word_counter = collections.Counter([word for sentence in spamham for word in sentence.split()])
    most_count = word_counter.most_common(30)
    # sorted data frame
    most_count = pd.DataFrame(most_count, columns=["Words", "Occurance"]).sort_values(by="Occurance", ascending = False)
    fig=px.bar(most_count, x = "Words", y = "Occurance", color="Occurance", template = 'ggplot2', title = title)
    fig.show()
```

```
word_count_plot(spamham_data["messages"],"Word Count Before Removing StopWords")
```



In this dataset, the word ‘to’ appears 2,226 times. Stopwords do not contribute meaningful information, so their analysis is unnecessary. Therefore, I removed both stopwords and punctuation to reduce the number of words. Since punctuation also does not serve an essential purpose in this context, the focus remains solely on the substantive terms present in the dataset.

To clean the data, I used the nltk library to remove stopwords and punctuation, then tokenized each word into character lists using split. I created a new column, "removed_stopwords," which contains all the processed characters.

```
#removing stopwords
def msg_process(msg):
    msg = msg.translate(str.maketrans('', '', string.punctuation))
    msg = [word for word in msg.split() if word.lower() not in stopwords.words('english')]
    return msg
```

```
spamham_data['removed_stopwords'] = spamham_data['messages'].apply(lambda row: msg_process(row))
```

```
spamham_data.head(5)
```

	catergories	messages	Length	no_of_words	removed_stopwords
0	ham	go until jurong point, crazy.. available only ...	111	24	[go, jurong, point, crazy, available, bugis, n...
1	ham	ok lar... joking wif u oni...	29	8	[ok, lar, joking, wif, u, oni]
2	spam	free entry in 2 a wkly comp to win fa cup fina...	155	37	[free, entry, 2, wkly, comp, win, fa, cup, fin...
3	ham	u dun say so early hor... u c already then say...	49	13	[u, dun, say, early, hor, u, c, already, say]
4	ham	nah i don't think he goes to usf, he lives aro...	61	15	[nah, dont, think, goes, usf, lives, around, t...

As shown in row 0, the word “until” has been excluded from the message. However, it should be noted that NLTK may not recognize every stopword; for example:

'u' = you
'im' = I’m
'2' =to
'ur' = your
'ill' = I’ll
'4' =for
'lor' = slang word
'r' =are
'n' = ands
'da' =slang word
'oh' =slang word
'dun' =slang word basically means don’t
'lar' =slang word
'den' = slang word
'hor' =slang word
'nah' =slang word

In this process, I compiled an additional list of stopwords and subsequently generated a column labelled removing_extra_stopwords.

```python
#Removing extra StopWords
remove_extra_stopwords = ['u', 'im', '2', 'ur', 'ill', '4', 'lor', 'r', 'n', 'da', 'oh', 'dun','lar','den','hor','nah']

spamham_data['removing_extra_stopwords'] = spamham_data['removed_stopwords'].apply(lambda msg: [word for word in msg if word not
```

```python
spamham_data.head(5)
```

	catergories	messages	Length	no_of_words	removed_stopwords	removing_extra_stopwords
0	ham	go until jurong point, crazy.. available only ...	111	24	[go, jurong, point, crazy, available, bugis, n...	[go, jurong, point, crazy, available, bugis, g...
1	ham	ok lar... joking wif u oni...	29	8	[ok, lar, joking, wif, u, oni]	[ok, joking, wif, oni]
2	spam	free entry in 2 a wkly comp to win fa cup fina...	155	37	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entry, wkly, comp, win, fa, cup, final,...
3	ham	u dun say so early hor... u c already then say...	49	13	[u, dun, say, early, hor, u, c, already, say]	[say, early, c, already, say]
4	ham	nah i don't think he goes to usf, he lives aro...	61	15	[nah, dont, think, goes, usf, lives, around, t...	[dont, think, goes, usf, lives, around, though]

For instance, in row 3, the words 'u', 'dun', and 'hor' were deleted.

After removing all stopwords, a new column called Final text was created to show the processed text, along with clean_length to indicate the change in length.

```
In [33]: # Joining clean text and adding new Length to the table
         def get_final_text(msg):
             final_text=" ".join([word for word in msg])
             return final_text
         spamham_data['final_text']=spamham_data['removing_extra_stopwords'].apply(lambda row : get_final_text(row))
         spamham_data['clean_length'] =spamham_data.final_text.str.len()
         spamham_data.head(10)
```

Out[33]:

	catergories	messages	Length	no_of_words	removed_stopwords	removing_extra_stopwords	final_text	clean_length
0	ham	go until jurong point, crazy.. available only ...	111	24	[go, jurong, point, crazy, available, bugis, n...	[go, jurong, point, crazy, available, bugis, g...	go jurong point crazy available bugis great wo...	80
1	ham	ok lar... joking wif u oni...	29	8	[ok, lar, joking, wif, u, oni]	[ok, joking, wif, oni]	ok joking wif oni	17
2	spam	free entry in 2 a wkly comp to win fa cup fina...	155	37	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entry, wkly, comp, win, fa, cup, final,...	free entry wkly comp win fa cup final tkts 21s...	133
3	ham	u dun say so early hor... u c already then say...	49	13	[u, dun, say, early, hor, u, c, already, say]	[say, early, c, already, say]	say early c already say	23
4	ham	nah i don't think he goes to usf, he lives aro...	61	15	[nah, dont, think, goes, usf, lives, around, t...	[dont, think, goes, usf, lives, around, though]	dont think goes usf lives around though	39
5	spam	freemsg hey there darling it's been 3 week's n...	148	39	[freemsg, hey, darling, 3, weeks, word, back, ...	[freemsg, hey, darling, 3, weeks, word, back, ...	freemsg hey darling 3 weeks word back id like ...	89

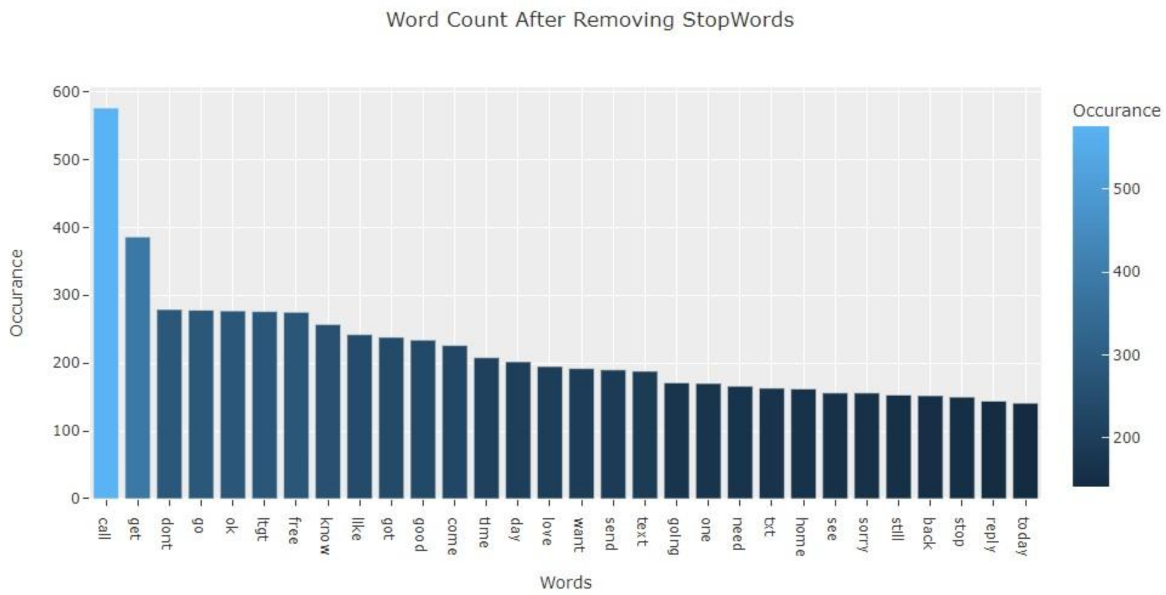
As demonstrated, the dataset has been reduced in length, with a total of 155,687 entries removed.

```
print("Original Length:",spamham_data.Length.sum())
print("Cleaned Length:",spamham_data.clean_length.sum())
print("Total Words Removed:",(spamham_data.Length.sum()) - (spamham_data.clean_length.sum()))
```

```
Original Length: 446422
Cleaned Length: 290735
Total Words Removed: 155687
```

The function word_count_plot was used to display the words after removing the stopwords. As shown, the number of words has decreased.

```python
#Word Count After removing StopWords and Extra StopWords
word_count_plot(spamham_data["final_text"],"Word Count After Removing StopWords")
```



WordCloud was generated to identify frequently occurring words and text within HAM and SPAM messages.

```python
#WordCloud used in HAM messages
ham_cloud = list(spamham_data.loc[spamham_data.catergories == 'ham', 'final_text'])

wordcloud_ham = WordCloud(width = 500,
                height = 500,
                background_color ='white', min_font_size = 9).generate(' '.join(ham_cloud))
plt.figure(figsize = (10, 9),dpi=50, facecolor = None)
plt.imshow(wordcloud_ham)
plt.axis("off")
plt.title('WordCloud for Ham message')
plt.tight_layout(pad = 0)
plt.show()
```

WordCloud: HAM Messages

```python
# Word Cloud in SPAM messages
spam_cloud = list(spamham_data.loc[spamham_data.catergories == 'spam', 'final_text'])

wordcloud_spam = WordCloud(width = 800, height = 800,
                    background_color ='white', min_font_size = 9).generate(' '.join(spam_cloud))
plt.figure(figsize = (10, 9),dpi=55, facecolor = None)
plt.imshow(wordcloud_spam)
plt.axis("off")
plt.title('WordCloud for Spam message')
plt.tight_layout(pad = 0)
plt.show()
```

WordCloud: SPAM Messages

Machine Learning: Building a classification model

After reviewing multiple articles on constructing classification models in machine learning, I opted to adopt the following procedure:
1) Transform textual data into vector representations
2) Partition the dataset into training and testing subsets
3) Utilise scikit-learn for model development
4) Fit the training data to the chosen model
5) Evaluate the classifier's performance

Text vectorization refers to the process of converting textual data into numerical vectors. This technique enables the extraction of meaningful information by representing all text in a form suitable for computational analysis. Several methods are commonly employed to achieve text vectorization:

- TF-IDF (Term Frequency-Inverse Document Frequency)
- Binary Term Frequency
- Count Vectorizer
- Bag of Words (BoW) Term Frequency
- Word2Vec

Several articles indicate that TF-IDF is effective for text analysis. TF-IDF calculates word frequency scores and helps to identify important words within a document. As this dataset consists solely of words, I will apply TF-IDF for text vectorization.

Prior to applying this process to the categories, "ham" was assigned a value of 0 and "SPAM" was assigned a value of 1.

```
#Replace "ham" to 0 and SPAM to 1
spamham_data = spamham_data.replace(['ham','spam'],[0, 1])
```

```
spamham_data.head(10)
```

	catergories	messages	Length	no_of_words	removed_stopwords	removing_extra_stopwords	final_text	clean_length
0	0	go until jurong point, crazy.. available only ...	111	24	[go, jurong, point, crazy, available, bugis, n...	[go, jurong, point, crazy, available, bugis, g...	go jurong point crazy available bugis great wo...	80
1	0	ok lar... joking wif u oni...	29	8	[ok, lar, joking, wif, u, oni]	[ok, joking, wif, oni]	ok joking wif oni	17
2	1	free entry in 2 a wkly comp to win fa cup fina...	155	37	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entry, wkly, comp, win, fa, cup, final,...	free entry wkly comp win fa cup final tkts 21s...	133

Convert the list of words to TF-IDF

```
#ML
#Vectorize to provide accuracy and precision
vectorizer = TfidfVectorizer(max_features=3000)
X = vectorizer.fit_transform(spamham_data['final_text'])
y=spamham_data['catergories'].values
```

Divide the dataset at random into a training set and a test set, with 20% of the data allocated for testing and the remainder for training.

```
#Train test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

Train a Multinomial Naive Bayes model and evaluate its accuracy, confusion matrix, and precision on the test set.

```
#Applying Naives Bayes Classifier Model
mnb = MultinomialNB()
```

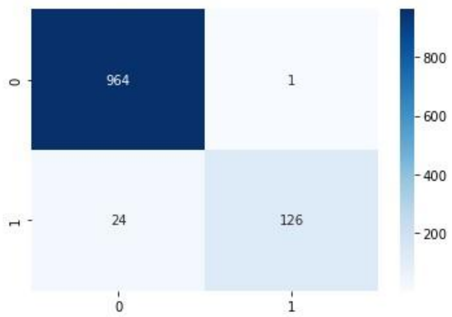
```
mnb.fit(X_train,y_train)
y_pred=mnb.predict(X_test)
print('Accuracy score of Multinomial NB is: ',accuracy_score(y_test,y_pred))
print('Confusion Matrix of Multinomial NB is: ',confusion_matrix(y_test,y_pred))
print('Precision score of the Multinomial NB is',precision_score(y_test,y_pred))
```

```
Accuracy score of Multinomial NB is:  0.9775784753363229
Confusion Matrix of Multinomial NB is:  [[964   1]
 [ 24 126]]
Precision score of the Multinomial NB is 0.9921259842519685
```

The Accuracy of this model is 0.97 and the precision is 0.99

```
#Classfication results of Cofusion Matrix
matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(matrix, annot = True, cmap='Blues', fmt = 'd')

# 964 ar ham(0) and 126 times it was spam (1)
#https://www.analyticsvidhya.com/blog/2021/09/performing-email-spam-detection-using-bert-in-python/
```

```
<AxesSubplot:>
```



The confusion matrix indicates that 964 messages were correctly classified as HAM, while 126 messages were accurately identified as SPAM. Additionally, 24 HAM messages were incorrectly labelled as SPAM, and one SPAM message was misclassified as HAM.

This is a demonstration intended to evaluate the classifier using various messages:

```
# Demo test model prediction
def test_classifier(sms):


    transformed = vectorizer.transform([sms])
    prediction =  mnb.predict(transformed)

    if prediction == 0 :
        return "This message is NOT spam!"
    else:
        return "This message is spam!"

print (test_classifier("mobile 11 months entitled update latest colour..."))
print (test_classifier("How are You?"))
print (test_classifier("free entry wkly comp win fa cup final tkts 21s..."))
print (test_classifier("Good morning Vincent"))
print (test_classifier("Free Entry"))
print (test_classifier("Urgent call this number now!"))
```

```
This message is spam!
This message is NOT spam!
This message is spam!
This message is NOT spam!
This message is spam!
This message is spam!
```

References

1) https://www.kaggle.com/code/karanchinchpure/spam-sms-email-classification-98accuracy/data?select=spam.csv -Dataset downloaded on: 10/8/2022

2) https://www.kaggle.com/code/iwasdata/spam-classification-using-multinomial-naive-bayes
3) https://www.kaggle.com/code/dilip990/spam-ham-detection-using-naive-bayesclassifier/notebook
4) https://www.kaggle.com/code/iwasdata/spam-classification-using-multinomial-naive-bayes
5) https://www.analyticsvidhya.com/blog/2021/06/automated-spam-e-mail-detectionmodelusing-common-nlp-tasks/
6) https://towardsdatascience.com/a-beginners-introduction-to-nlp-building-a-spam-classifiercf0973c7f42c
7) https://medium.com/@insight_imi/sms-spam-classification-using-na%C3%AFve-bayesclassifier-780368549279
8) https://medium.com/mlearning-ai/build-a-spam-classifier-in-python-25e1511e954