SPAM MAIL DETECTION

PHASE 1

S Abhishek

AM.EN.U4CSE19147

TABLE OF CONTENTS

1

- **Data Preprocessing**
  - ❖ **Find** the columns with **only Null** values
  - ❖ **Drop** the columns with **only Null** values
  - ❖ **Find** the rows with **any Null** values
  - ❖ **Drop** the rows with **any Null** values
  - ❖ **Drop** the **Duplicate** rows

2

- **Data Summarization**
  - ❖ Descriptive Statistics

3

- **Text Preprocessing**
  - ❖ Adding the **text Length** Column for each record
  - ❖ Word **Tokenization**
  - ❖ Finding **Mean Word Length**
  - ❖ Removing **Punctuations** and **Stop Words**

4	• **Data Visualization** ❖ **Seaborn Heat Map** for the graphical representation of missing data ❖ **Msno Bar Graph** for the simple visualization of nullity by column ❖ **Msno Heat Map** for visualizing the correlation between missing values of different columns ❖ **Pyplot** the length of Spam & Ham Texts ❖ **Distplot** the Spam & Ham record’s length after tokenizing ❖ **Distplot** the Mean Word Length ❖ **Distplot** the distribution of Stop Words Ratio ❖ **Countplot** the Spam & Ham ratio ❖ **Word Cloud** Visualization
5	• **Data Interpretation**

Data Pre-processing

Find the columns with **only Null** values

```
# Find the Number of Rows that has Nan Value in it

data.isnull().sum()

text    6

spam    8

dtype: int64


# Count the No of Non-NA cells for each column or row

data.count()

text    11300

spam    11298

dtype: int64


# Find the Number of Rows that has Nan Value in it

Null_Data = data.isnull().sum()


# List for storing the Null Column Names

Null_Columns = []
```

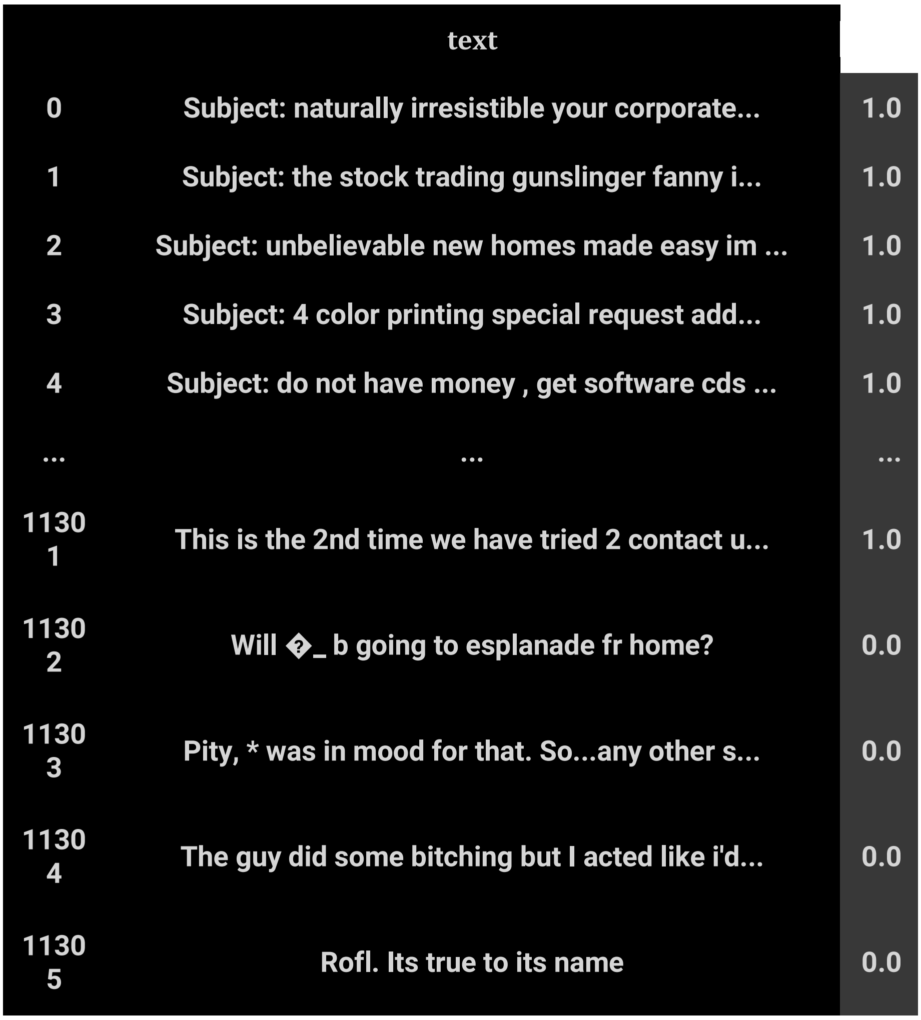
```
for i in range(len(Null_Data)):

# If the number of Null Values in the Row is equal to the total number of Records, then it means that the whole column contains Null value in it.

if Null_Data[i] == Rows - 1 or Null_Data[i] == Rows:

    Null_Columns.append(Column_Names[i])

# Print all Columns which has only NULL values

print(Null_Columns)
```

Output : []

- It's evident that there is no column in the dataset which has only NULL values.

Drop the columns with **only Null** values

```
# Delete all NULL Columns which has only NULL values

for i in Null_Columns:
  del data[i]
data
```



	text	
0	Subject: naturally irresistible your corporate...	1.0
1	Subject: the stock trading gunslinger fanny i...	1.0
2	Subject: unbelievable new homes made easy im ...	1.0
3	Subject: 4 color printing special request add...	1.0
4	Subject: do not have money , get software cds ...	1.0
...	...	...
11301	This is the 2nd time we have tried 2 contact u...	1.0
11302	Will �_ b going to esplanade fr home?	0.0
11303	Pity, * was in mood for that. So...any other s...	0.0
11304	The guy did some bitching but I acted like i'd...	0.0
11305	Rofl. Its true to its name	0.0

Find the rows with **any Null** values

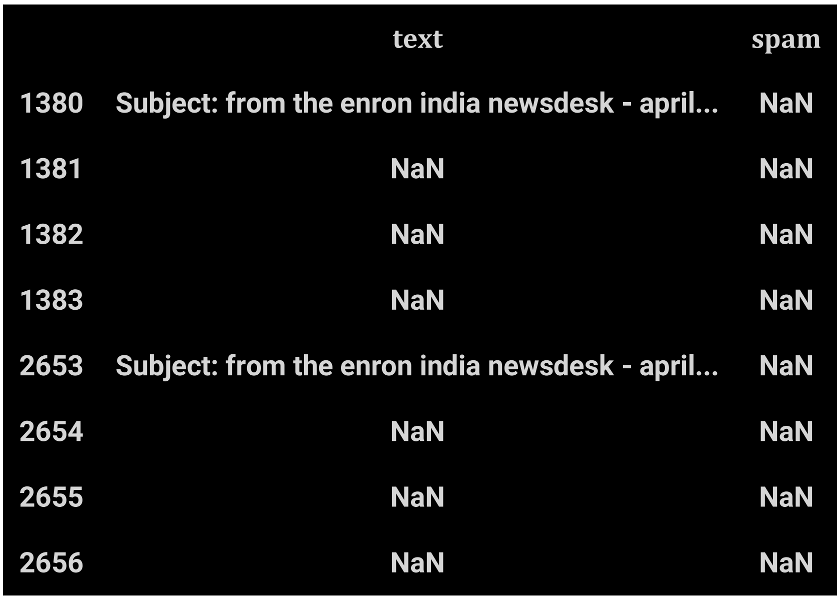
data.isnull().any()

```
text    True
spam    True
dtype: bool
```

data.isnull().sum()

```
text    6
spam    8
dtype: int64
```

```
# Display the Rows which has one or more NULL values in it

data[data.isnull().any(axis=1)]
```



	text	spam
1380	Subject: from the enron india newsdesk - april...	NaN
1381	NaN	NaN
1382	NaN	NaN
1383	NaN	NaN
2653	Subject: from the enron india newsdesk - april...	NaN
2654	NaN	NaN
2655	NaN	NaN
2656	NaN	NaN

Drop the rows with **any Null** values

```
data.dropna(inplace=True)

data.isnull().any()
```

```
text    False
spam    False
dtype: bool
```

```
print(data.isnull().sum())
```

```
text    0
spam    0
dtype: int64
```

Drop the **Duplicate** rows

```
data.shape
```

```
(11298, 2)
```

```
# Check if there is any Duplicate Rows
duplicate = data[data.duplicated()]
print("Number of Duplicate rows: ", duplicate.shape)
```

```
Number of Duplicate rows:  (436, 2)
```

```
data.count()
```

```
text    11298
spam    11298
dtype: int64
```

```
# Drop all the Duplicate Rows
data = data.drop_duplicates()
data.count()
```

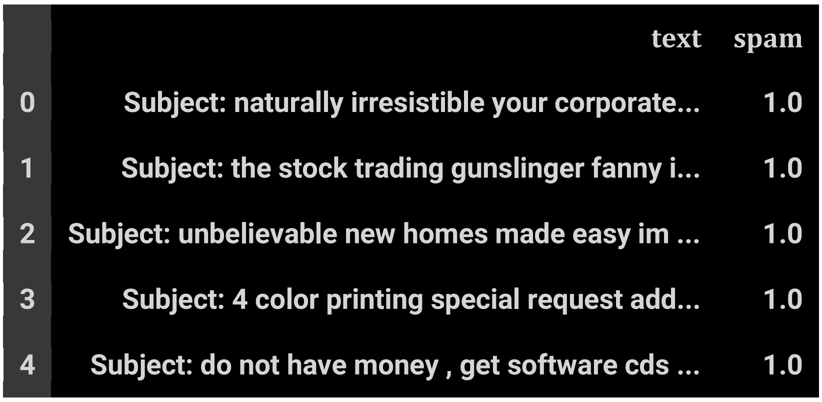
```
text    10862
spam    10862
dtype: int64
```

Data Summarization

Descriptive Statistics

- Descriptive statistics analysis helps to describe the basic features of dataset and obtain a brief summary of the data.
- The describe() method in Pandas library helps us to have a brief summary of the dataset.
- It automatically calculates basic statistics for all numerical variables excluding NaN (we will come to this part later) values.

```
# Display First 5 Records
data.head()
```

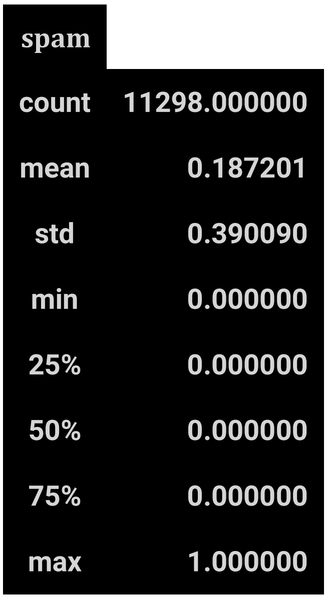


	text	spam
0	Subject: naturally irresistible your corporate...	1.0
1	Subject: the stock trading gunslinger fanny i...	1.0
2	Subject: unbelievable new homes made easy im ...	1.0
3	Subject: 4 color printing special request add...	1.0
4	Subject: do not have money , get software cds ...	1.0

```
# The info() function is used to print a concise summary of Data Frame.

data.info()
```

```
RangeIndex: 11306 entries, 0 to 11305
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   text    11300 non-null  object 
 1   spam    11298 non-null  float64
dtypes: float64(1), object(1)
memory usage: 176.8+ KB
```

```
# Pandas describe() is used to view some basic statistical details like percentile,
mean, std etc. of a data frame or a series of numeric values.

data.describe()
```



	spam
count	11298.000000
mean	0.187201
std	0.390090
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

```
# The dtypes property is used to find the dtypes in the DataFrame.

data.dtypes
```

```
text     object
spam    float64
dtype: object
```

```
# No of Rows

Rows = data.shape[0]


# No of Columns

Columns = data.shape[1]


print("Rows :", Rows)
print("Columns :", Columns)


# Column Names

Column_Names = data.columns
```

```
Rows : 11306

Columns : 2
```

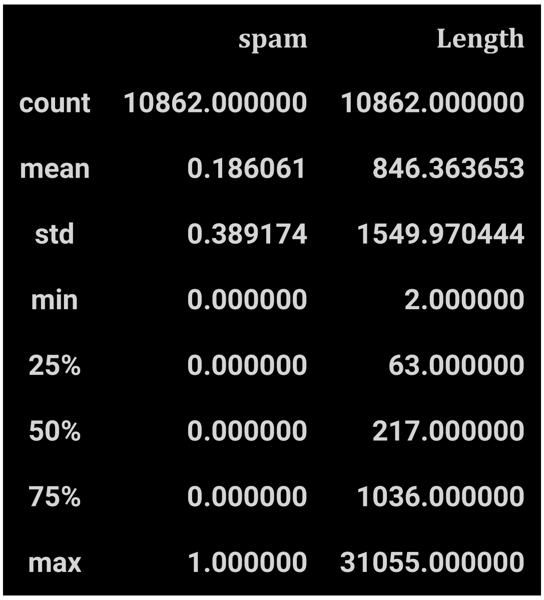
Text Preprocessing

Adding the **text Length** Column for each record

```
# Store the Length of the messages in the New Column with respective to each of the records


data['Length'] = data['text'].apply(len)
data['Length'].max()
```

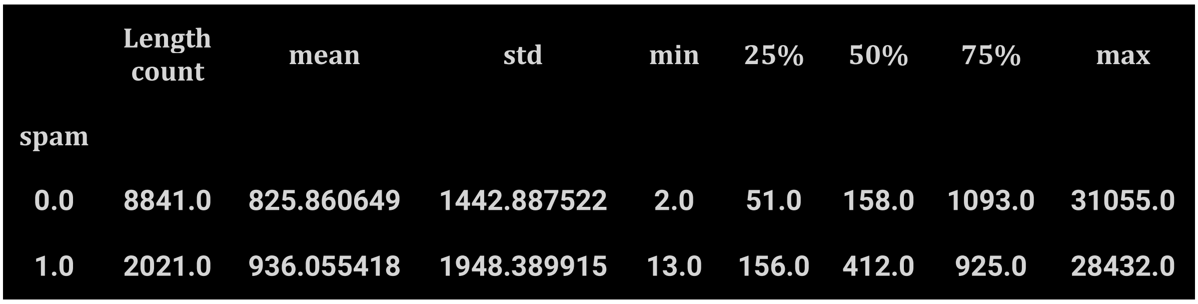
```
31055
```

```
data.describe()
```



	spam	Length
count	10862.000000	10862.000000
mean	0.186061	846.363653
std	0.389174	1549.970444
min	0.000000	2.000000
25%	0.000000	63.000000
50%	0.000000	217.000000
75%	0.000000	1036.000000
max	1.000000	31055.000000

```
# See the different classes of values in the Spam Column

data.groupby('spam').describe()
```



	Length count	mean	std	min	25%	50%	75%	max
spam								
0.0	8841.0	825.860649	1442.887522	2.0	51.0	158.0	1093.0	31055.0
1.0	2021.0	936.055418	1948.389915	13.0	156.0	412.0	925.0	28432.0

Word **Tokenization**

```
# Count the max word length used in any spam or ham email.

# Import NLTK Library


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

```
from nltk.tokenize import word_tokenize

# Finding the length of all Ham & Spam texts

Ham_Words_Length = [len(word_tokenize(title)) for title in data[data['spam']==0].text.values]

Spam_Words_Length = [len(word_tokenize(title)) for title in data[data['spam']==1].text.values]

print("\nHam Words Length :", max(Ham_Words_Length))

print("\nSpam Words Length :", max(Spam_Words_Length))

# Check which has the highest length

if max(Ham_Words_Length) > max(Spam_Words_Length):

  print("\nHam Text Length is Larger")

else:

  print("\nSpam Text Length is Larger")
```

```
Ham Words Length : 6350

Spam Words Length : 6131

Ham Text Length is Larger
```

- For ham email, the maximum number of ham words used in an email is 6350.
- For spam email, the maximum number of spam words used in an email is 6131.
- It's evident that the spam emails have less words as compared to ham emails.

Finding **Mean Word** Length

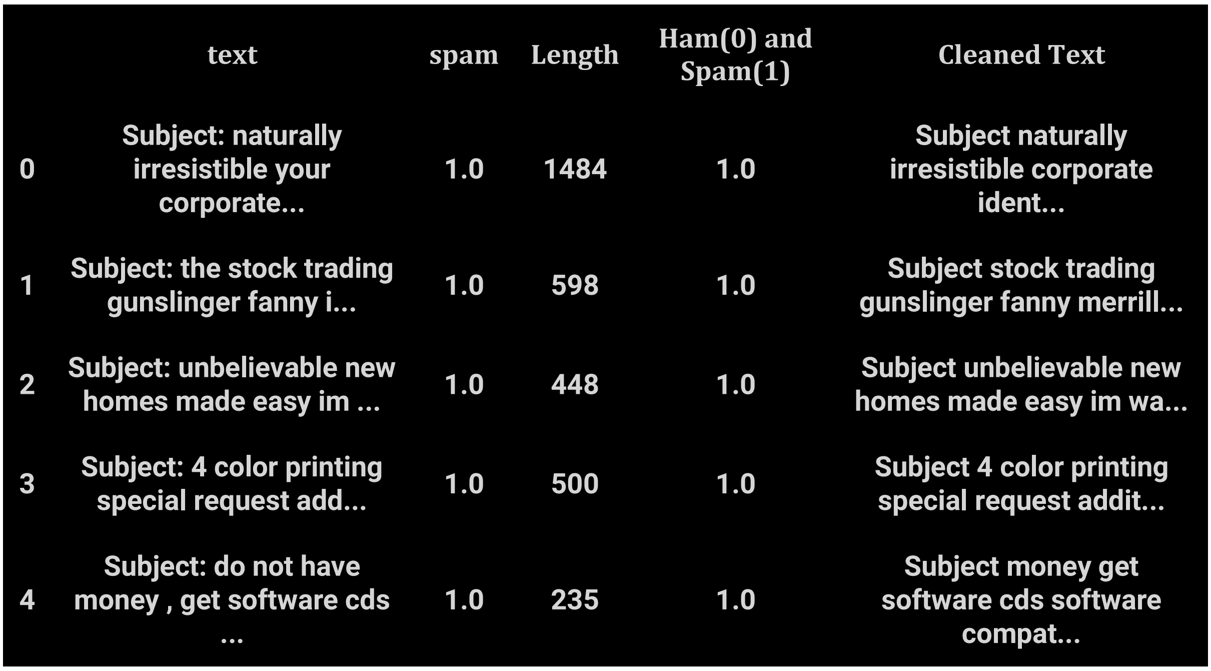
```
import numpy as np

# Function to find the Mean Word Length
def Mean_Word_Length(x):
    length = np.array([])
    for word in word_tokenize(x):
        length = np.append(length, len(word))
    return length.mean()

Ham_Meanword_Length =
data[data['spam']==0].text.apply(Mean_Word_Length)

Spam_Meanword_Length =
data[data['spam']==1].text.apply(Mean_Word_Length)
```

Removing **Punctuations** and **Stop Words**

- Stop Words are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc).
- They don't add much information to the text.
- Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.
- Stop words are available in abundance in any human language.
- By removing these words, we remove the low-level information from our text in order to give more focus to the important information.
- In order words, we can say that the removal of such words does not show any negative consequences on the model we train for our task.
- Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.
- We do not always remove the stop words. The removal of stop words is highly dependent on the task we are performing and the goal we want to achieve.
- For example, if we are training a model that can perform the sentiment analysis task, we might not remove the stop words.
- Movie review: “The movie was not good at all.” Text after removal of stop words: “movie good”.
- We can clearly see that the review for the movie was negative.
- However, after the removal of stop words, the review became positive, which is not the reality.

- Thus, the removal of stop words can be problematic here. Tasks like text classification do not generally need stop words as the other words present in the dataset are more important and give the general idea of the text.
- So, we generally remove stop words in such tasks.

```
import string
class Data_Clean():
    def __init__(self):
        pass
    def Message_Cleaning(self, message):
      Text = [char for char in message if char not in string.punctuation]
      Text = ''.join(Text)
      Text_Filtered = [word for word in Text.split() if word.lower() not in stopwords.words('english')]
      Text_Filtered = ' '.join(Text_Filtered)
      return Text_Filtered

    def Clean(self, U_data):
      C_Data = U_data.apply(self.Message_Cleaning)
      return C_Data
Cleaned_Data = Data_Clean()

data['Cleaned Text'] = Cleaned_Data.Clean(data['text'])

data.head()
```



	text	spam	Length	Ham(0) and Spam(1)	Cleaned Text
0	Subject: naturally irresistible your corporate...	1.0	1484	1.0	Subject naturally irresistible corporate ident...
1	Subject: the stock trading gunslinger fanny i...	1.0	598	1.0	Subject stock trading gunslinger fanny merrill...
2	Subject: unbelievable new homes made easy im ...	1.0	448	1.0	Subject unbelievable new homes made easy im wa...
3	Subject: 4 color printing special request add...	1.0	500	1.0	Subject 4 color printing special request addit...
4	Subject: do not have money , get software cds ...	1.0	235	1.0	Subject money get software cds software compat...

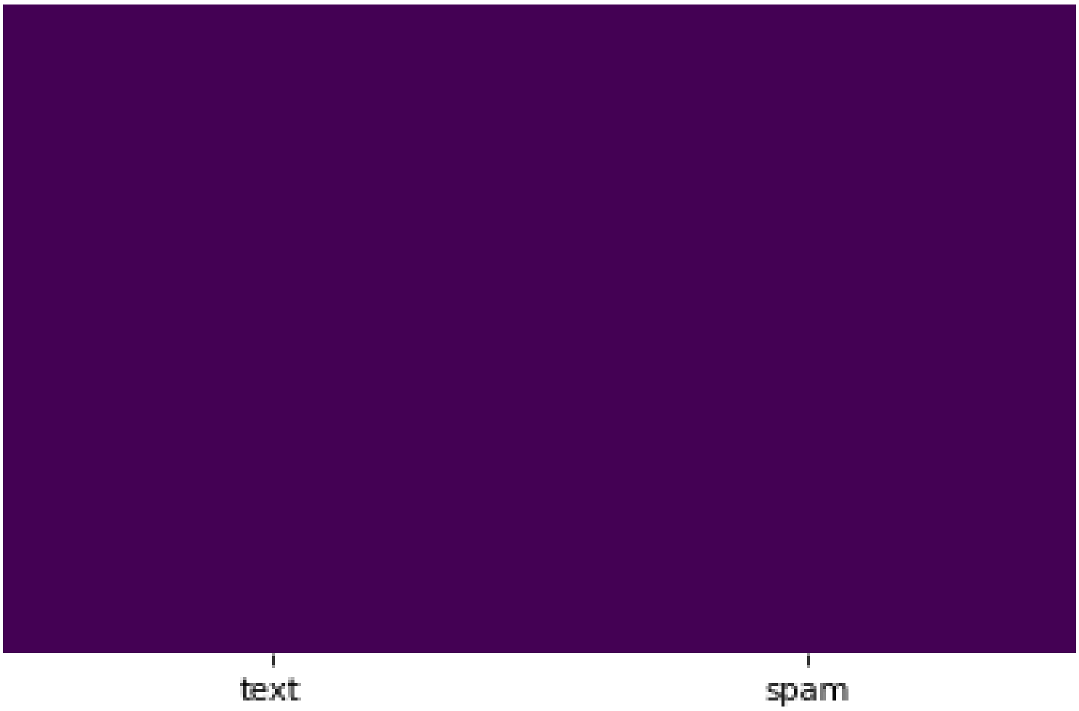
Data Visualization

Seaborn Heat Map for the graphical representation of missing data

- Heatmaps visualize the data in a 2-dimensional format in the form of coloured maps.
- The colour maps use hue, saturation, or luminance to achieve colour variation to display various details.
- This colour variation gives visual cues to the readers about the magnitude of numeric values.
- Heat Maps is about replacing numbers with colours because the human brain understands visuals better than numbers, text, or any written data.

- Heatmaps can describe the density or intensity of variables, visualize patterns, variance, and even anomalies.
- Heatmaps show relationships between variables.
- These variables are plotted on both axes. We look for patterns in the cell by noticing the colour change.

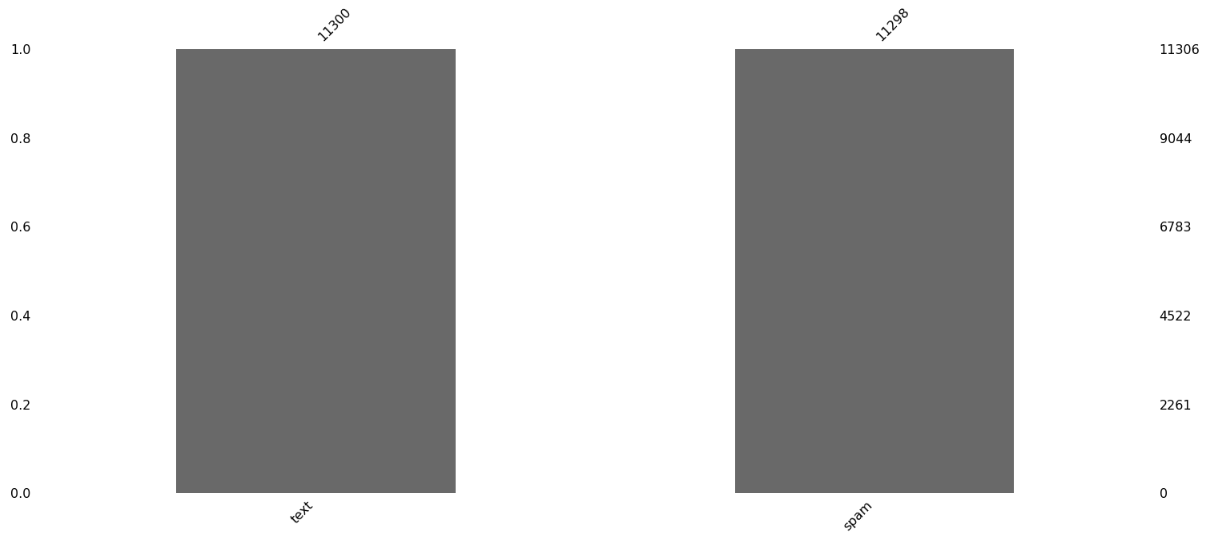
```
# To Check missing value
Import Seaborn
import seaborn as sn
# Heat Map Visualization
sn.heatmap(data.isnull(), cbar=False, yticklabels=False, cmap='viridis')
```



Msno Bar Graph for the simple visualization of nullity by column

- Pandas provides functions to check the number of missing values in the dataset.
- Missingno library takes it one step further and provides the distribution of missing values in the dataset by informative visualizations.
- Using the plots of missingno, we are able to see where the missing values are located in each column and if there is a correlation between missing values of different columns.
- Before handling missing values, it is very important to explore them in the dataset.

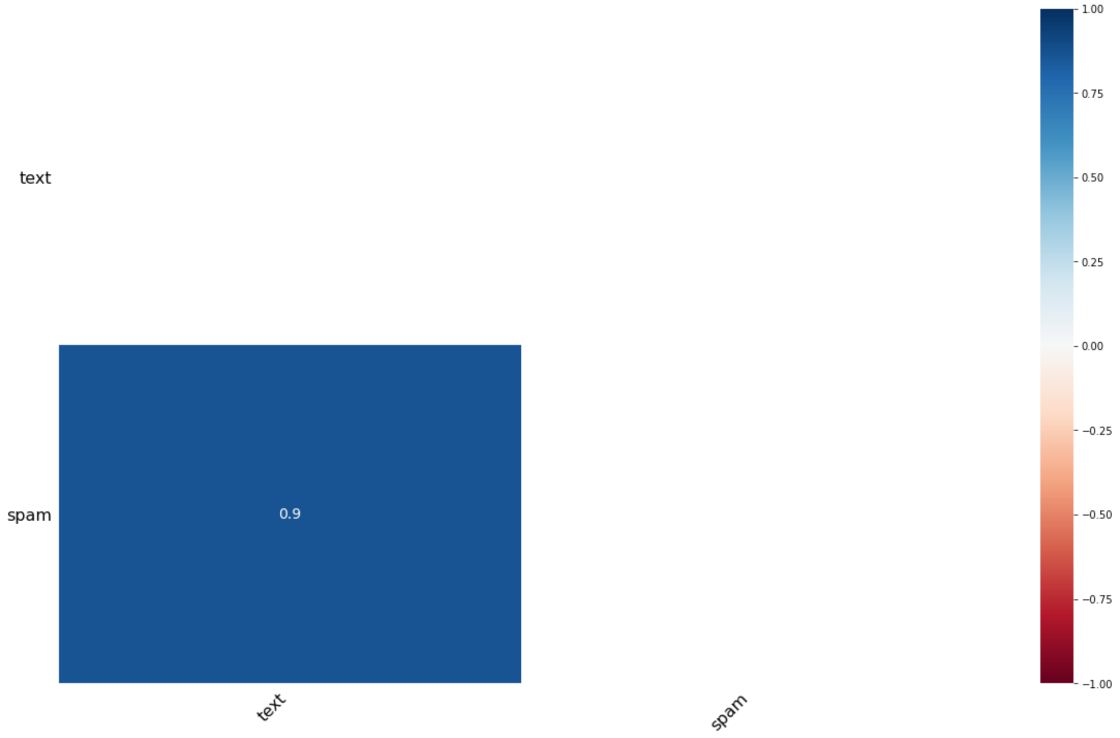
```
# Import missingno Library
import missingno as msno

# Plot the Bar Graph
msno.bar(data)
```



Msno Heat Map for visualizing the correlation between missing values of different columns

# Plot the Heat Map

msno.heatmap(data)

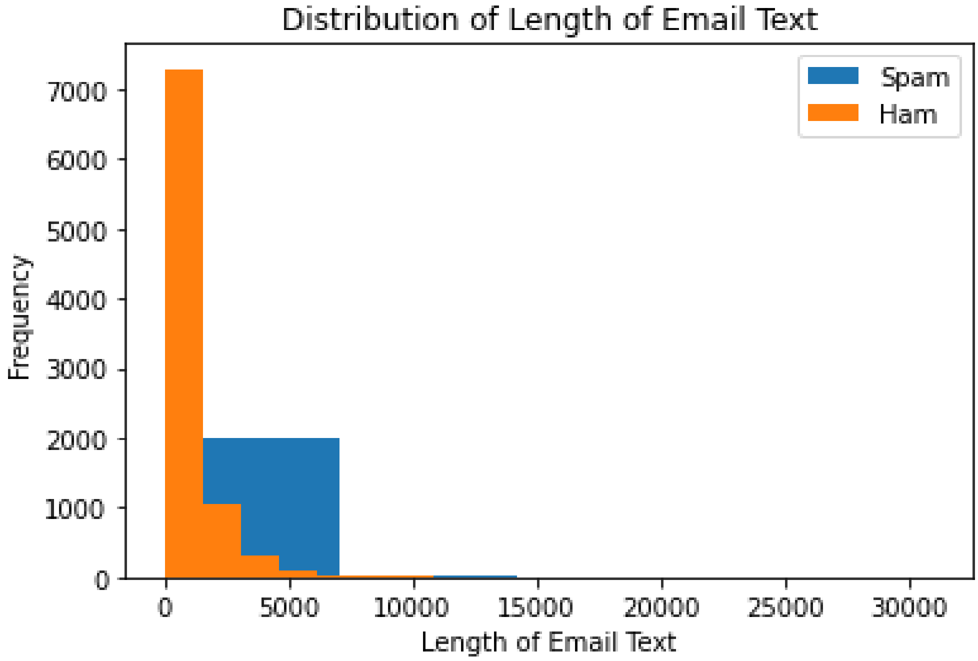


Pyplot the length of Spam & Ham Texts

- A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent.
- The bar plots can be plotted horizontally or vertically.
- A bar chart describes the comparisons between the discrete categories.

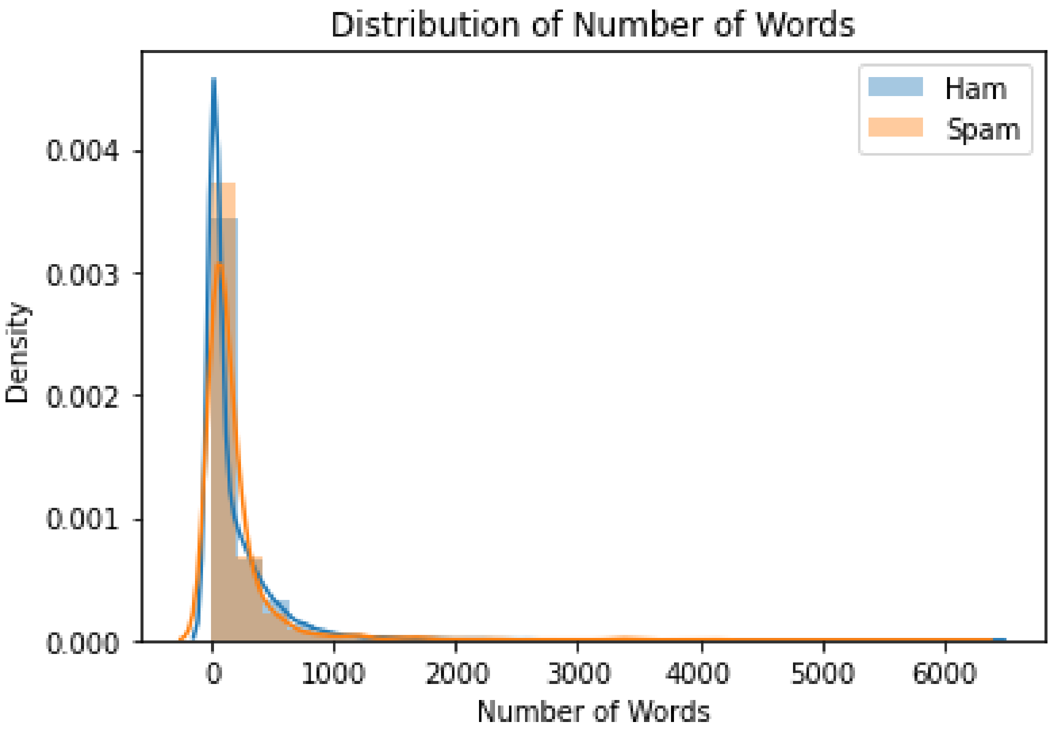
- One of the axes of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

```
# Import Matplotlib Library
import matplotlib.pyplot as plt

# Split the Spam & Ham Records
Spam_Length =  data[data['spam']==1]
Ham_Length =  data[data['spam']==0]

# Plot the Length of Spam & Ham Messages
Spam_Length['Length'].plot(bins=4, kind='hist',label = 'Spam')
Ham_Length['Length'].plot(bins=20, kind='hist',label = 'Ham')
plt.title('Distribution of Length of Email Text')
plt.xlabel('Length of Email Text')
plt.legend()
```



Distplot the Spam & Ham record’s length after tokenizing

```
ax = sn.distplot(Ham_Words_Length, norm_hist = True, bins = 30, label = 'Ham')

ax = sn.distplot(Spam_Words_Length, norm_hist = True, bins = 30, label = 'Spam')

print()

plt.title('Distribution of Number of Words')

plt.xlabel('Number of Words')

plt.legend()

plt.show()
```

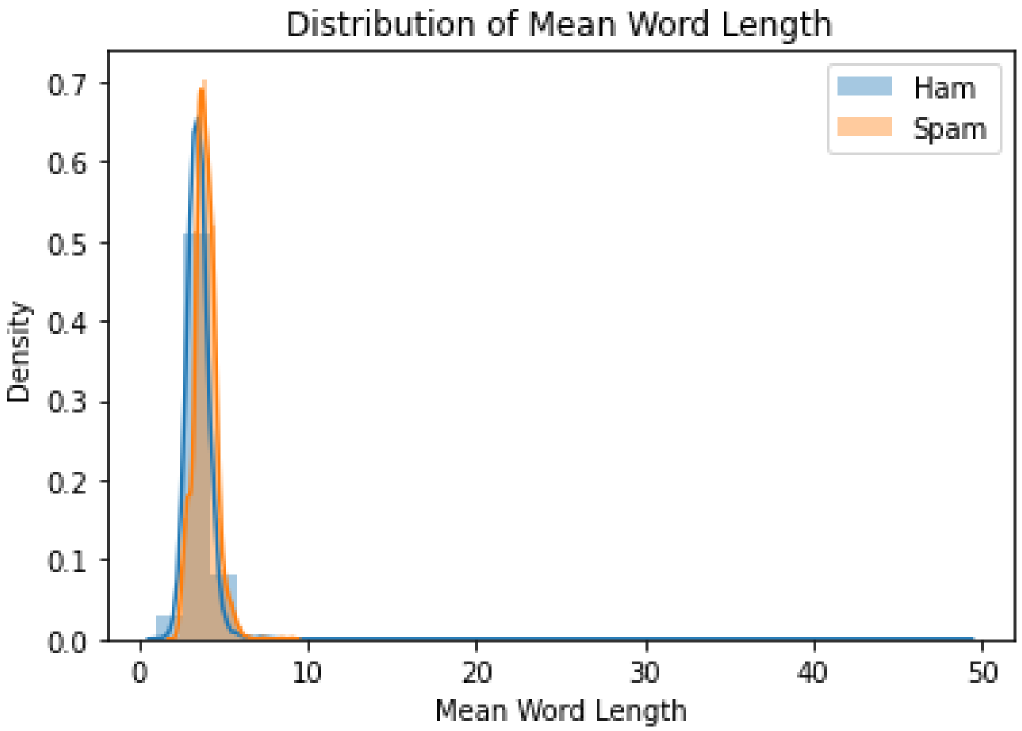


Distplot the Mean Word Length

- ❖ A Distplot or distribution plot, depicts the variation in the data distribution.

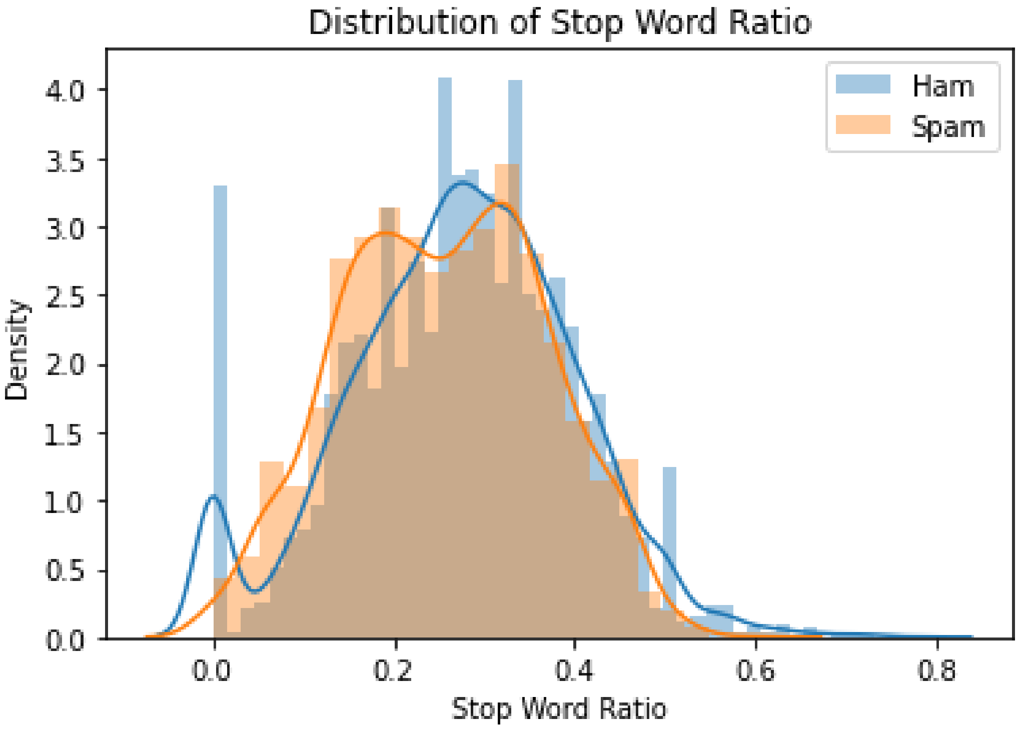
- ❖ Seaborn Distplot represents the overall distribution of continuous data variables.
- ❖ The Seaborn module along with the Matplotlib module is used to depict the distplot with different variations in it.
- ❖ The Distplot depicts the data by a histogram and a line in combination to it.

```
# Plot the Graph of Distribution of the Mean Word Length

sn.distplot(Ham_Meanword_Length, norm_hist = True, bins = 30, label = 'Ham')

sn.distplot(Spam_Meanword_Length , norm_hist = True, bins = 30, label = 'Spam')

print()

plt.title('Distribution of Mean Word Length')

plt.xlabel('Mean Word Length')

plt.legend()

plt.show()
```



Distplot the distribution of Stop Words Ratio

```
ham_stopwords = data[data['spam']==0].text.apply(stop_words_ratio)

spam_stopwords = data[data['spam']==1].text.apply(stop_words_ratio)

sn.distplot(ham_stopwords, norm_hist = True, label = 'Ham')

sn.distplot(spam_stopwords,  label = 'Spam')

plt.title('Distribution of Stop Word Ratio')

plt.xlabel('Stop Word Ratio')

plt.legend()

plt.show()
```

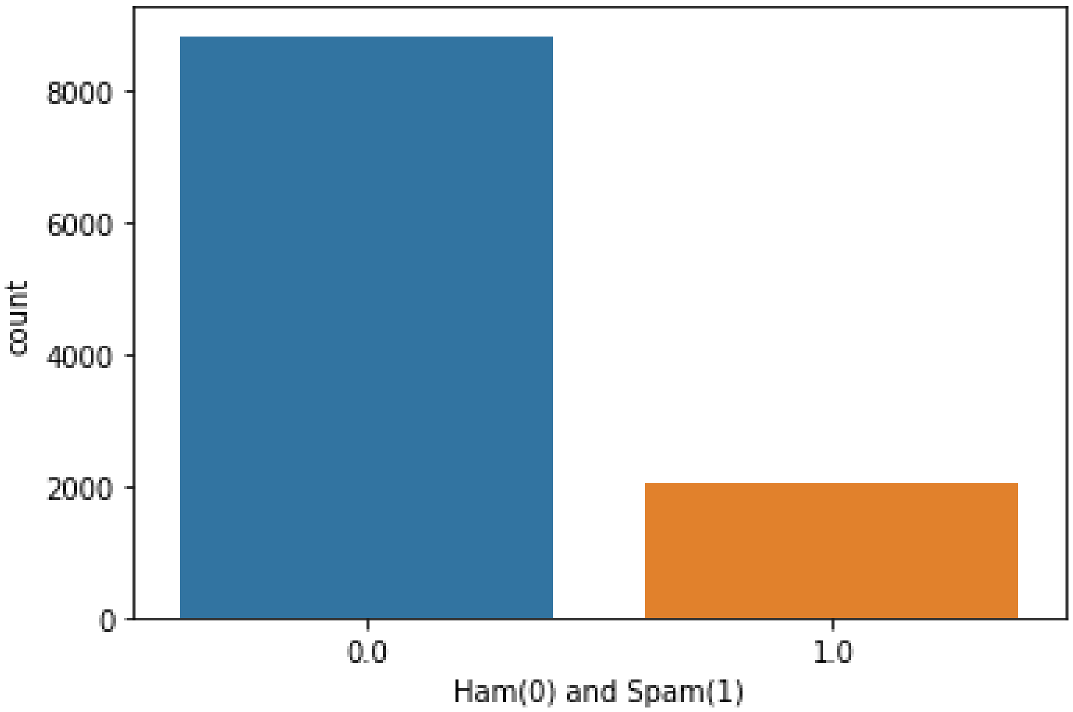


Countplot the Spam & Ham ratio

- ❖ The countplot is used to represent the occurrence(counts) of the observation present in the categorical variable.

- It uses the concept of a bar chart for the visual depiction.
- To construct a histogram, the first step is to “bin”, divide the entire range of values into a series of intervals—and then count how many values fall into each interval.
- The bins are usually specified as consecutive, non-overlapping intervals of a variable.
- The bins (intervals) must be adjacent and are often (but are not required to be) of equal size.
- The x-axis of the histogram denotes the number of bins while the y-axis represents the frequency of a particular bin.
- The number of bins is a parameter which can be varied based on how you want to visualize the distribution of your data.

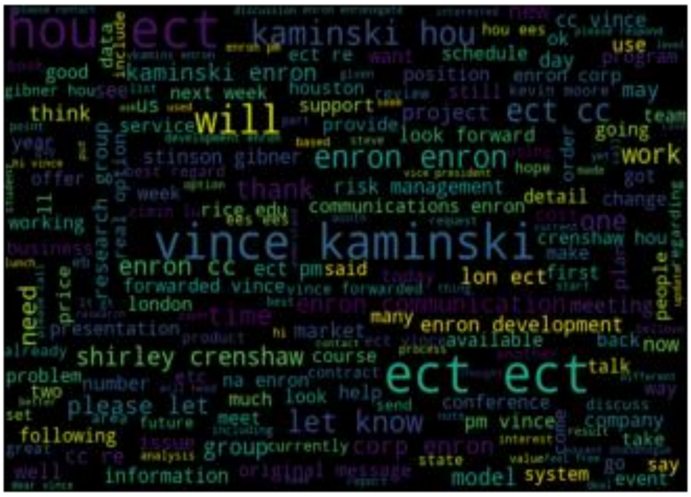
```
# Divide the messages into spam and ham

ham = data.loc[data['spam']==0]

spam = data.loc[data['spam']==1]

spam['Length'].plot(bins=60, kind='hist')

data['Ham(0) and Spam(1)'] = data['spam']

sn.countplot(data['Ham(0) and Spam(1)'], label = "Count")
```



Word Cloud Visualization

- Word cloud is a technique for visualising frequent words in a text where the size of the words represents their frequency.
- A word cloud (also called tag cloud or weighted list) is a visual representation of text data. Words are usually single words, and the importance of each is shown with font size or color.
- Python fortunately has a wordcloud library allowing to build them.
- The wordcloud library is here to help you build a wordcloud in minutes using the WordCloud() Library.

```
class Word_Cloud():
    def __init__(self):
        pass
    def variance_column(self, data):
        return variance(data)
    def word_cloud(self, data_frame_column, output_image_file):
        text = " ".join(review for review in data_frame_column)
        stopwords = set(STOPWORDS)
        stopwords.update(["subject"])
        wordcloud = WordCloud(width = 1200, height = 800, stopwords=stopwords, max_font_size = 90, margin=0, background_color = "black").generate(text)
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.show()
        wordcloud.to_file(output_image_file)
        return

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
word_cloud = Word_Cloud()
word_cloud.word_cloud(ham["text"], "Ham.png")
word_cloud.word_cloud(spam["text"], "Spam.png")
```



Data Interpretation

- ❖ Original Data Set : GitHub
- ❖ Processed Data Set : GitHub

Thankyou!!