SPAM MAIL DETECTION

PHASE 2

S Abhishek

AM.EN.U4CSE19147

Contents

- **Text Preprocessing**
    - Adding the text Length Column for each record
    - Word Tokenization
    - Finding Mean Word Length
    - Removing Punctuations and Stop Words
    - Porter Stemmer
- **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
    - Pieplot the Spam & Ham ratio
    - Histplot the number of characters, words and sentences

ABSTRACT

Spam email is one of the most demanding and troublesome internet issues in today's world of communication and technology. Sending malicious link through spam emails which can harm our system and can also seek in into your system. Spam emails not only influence the organizations financially but also exasperate the individual email user. So, it is needed to Identify those spam mails which are fraud and this project will identify those spam by using techniques of machine learning, where it applies algorithms on our data sets and the best algorithm is selected for the email spam detection having best precision and accuracy.

Ill Posed Problem:

- I need to classify the spam and non-spam mails.

Well Posed Problem:

- **Task** – Classifying emails as spam or not
- **Performance Measure** – The fraction of emails accurately classified as spam or not spam
- **Experience** – Observing you label emails as spam or not spam.

Introduction

Motivation

- In our day-to-day life, Spam Emails are considered to be annoying and repetitive, which is solely send for the purpose of advertisement and brand promotion.
- People are using them for illegal and unethical conducts, phishing and fraud.
- Sending malicious link through spam emails which can harm our system and can also seek in into your system.
- Creating a fake profile and email account is much easy for the spammers, they pretend like a genuine person in their spam emails, these spammers target those peoples who are not aware about these frauds.
- Although we block such emails, it is of no use as spam emails are still prevalent.
- Thus, we need to build a robust real-time email spam classifier that can efficiently and correctly flag the incoming mail spam, as either a spam or ham (non – spam) email.

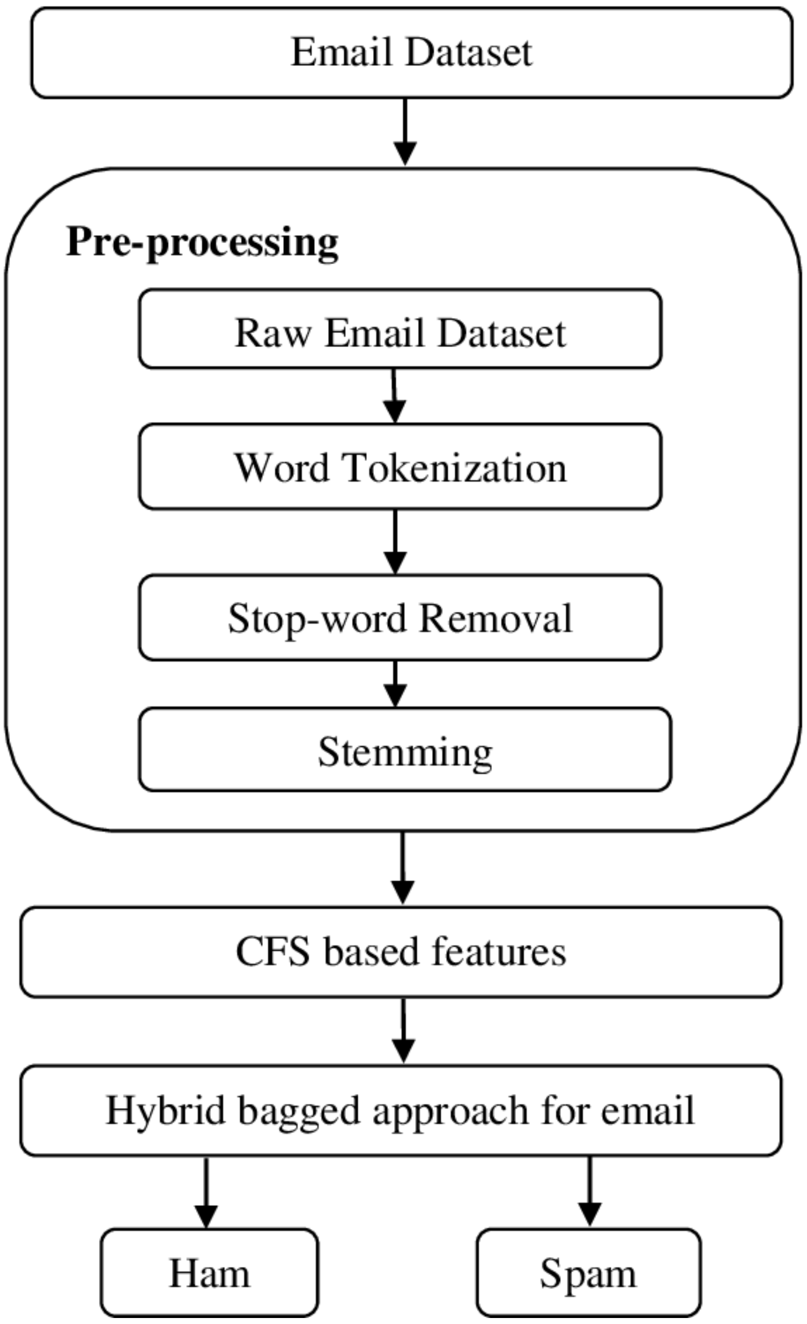
Benefits

- Spam filters can provide a great firewall to the spam emails which can be carriers of dangerous computer viruses.
- Spam filter that blocks spam emails from reaching the inbox can save all important data.
- Spam filters also saves time. Business employees do not have to go through numerous emails to decide which ones are spams, as sometimes that can be hard to decide. The time saved can be used to increase productivity.
- Spam filters can help keep a company maintains its reputation. They can block viruses from reaching consumers data and prevent any spam mail accidentally being forwarded to consumers.
- Spam filters protect the servers from being overloaded with non-essential emails, and the worse problem of being infected with spam software that may turn them into spam servers themselves.

Solution Use

- Private companies, who have their own email servers, want their data to be more secure. In such cases, email spam classification solutions can be provided to them.

- Also, Employees of the company need not go through each and every email, and can sort out the spam emails from the list. Thus, the time saved can be used to increase productivity
- While this may sound like a straightforward task, it can be a challenge for filters that are not constantly updated according to the most recent spam techniques and senders.
- Spammers may change the address from which emails come or the wording inside the header or body to bypass out-of-date spam filters.
- This can be effective if the spam filter is not updated with the correct information on a regular basis.
- It is important to make sure that the spam filter has adequate spam intelligence.
- If it does, it can block hundreds or thousands of spam emails every month.



Functional Requirements

- The main function of this project is to clarify the e-mails which is done by first taking out the feature vector extraction which involves first taking out whether the word is a spam or not.

Non-Functional Requirements

- Ensures high availability of email datasets.
- User should get the results as fast as possible.
- It should be easy to use (ie) user is just required to type the words and click, then the result is displayed or the user is required to enter a pair of reasonable sentences.

DATASET FINALIZATION

Data Set 1	Data Set 2	Data Set 3	Data Set 4

- These datasets consist of emails sent mostly by the senior management of the Enron Corporation which contains most of the words or phrases that are particularly common in spam e-mails which are unprocessed/Unorganized.
- The dataset consists of 30207 emails of which 16545 emails are labelled as ham and 13662 emails are labelled as spam.
- Before using the data set for pre-processing it has to be organized with only useful information.

- In this experiment we are using a processed version of this dataset specifically made for spam and ham classification.

Features in the datasets

- There are around 6000 entries in each dataset approximately.
- There are **4** attributes in the data set.
    - Text
    - Spam
    - Length
    - Clean Text
- **Text**
    - This attribute contains the contents of email from various organizations along with the date, time, subject and the message.
- **Spam**
    - This attribute contains the classification of the emails whether it is spam/ham.
        - **1** – spam
        - **0** – ham

    - Using the key words related to spam mails, the received mails are classified as spam and not spam.
- **Length**
    - This attribute contains the count of characters in the email.
- **Clean Text**
    - This attribute contains the processed text, after removing all unnecessary characters like "**: ,**" etc.
- The Enron Email Dataset was used in Corpus Linguistics and language analysis for email search and Expert search.
- Tech Giant use this data set to analyse the data from email analytics dashboard and also to compare it to the goals and the KPIs the company has set.
- Social media marketing companies use this data set to improve their email marketing results.

Assumptions

- Some of the datasets only has the content of the mail, and its not classified into spam and not spam (ie) Missing of attributes that are used to specify whether the mail belongs to spam or non spam.

- All the available datasets are not updated for a long period and thus they are not capable of identifying the most recent spam techniques and senders.
- We assume that, because of the non updation of the dataset over a period of time, we may miss all new spam keywords which is used to classify the spam and non-spam mails.

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
```

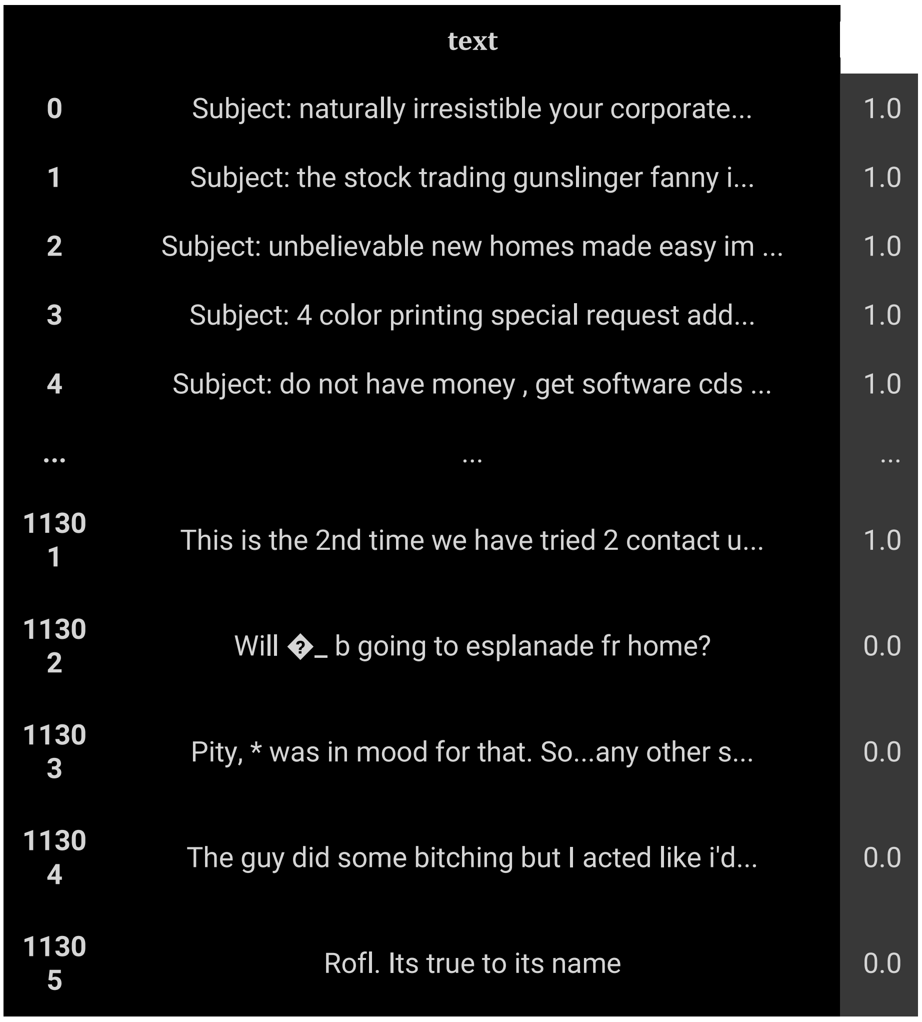
```
# Find the Number of Rows that has Nan Value in it

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 Rec
ords, 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

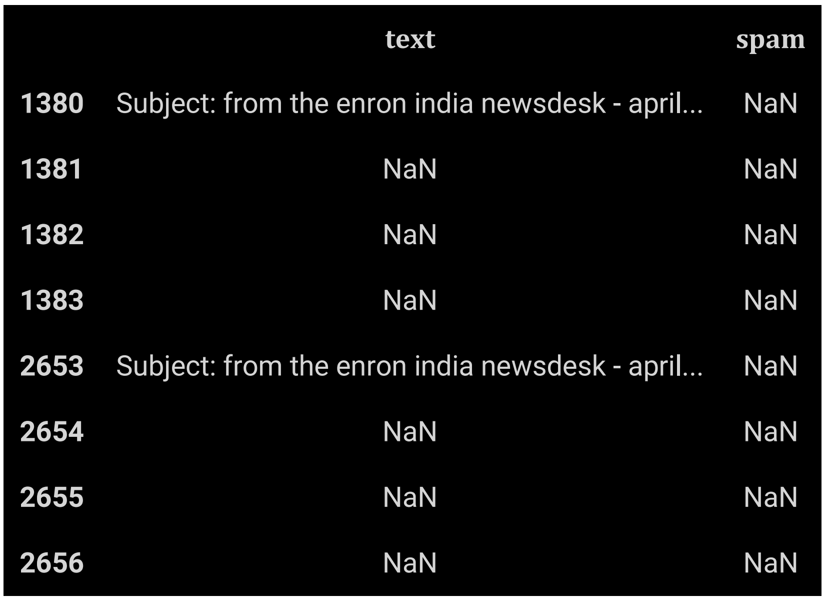
```
data.isnull().any()
```

```
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()
```

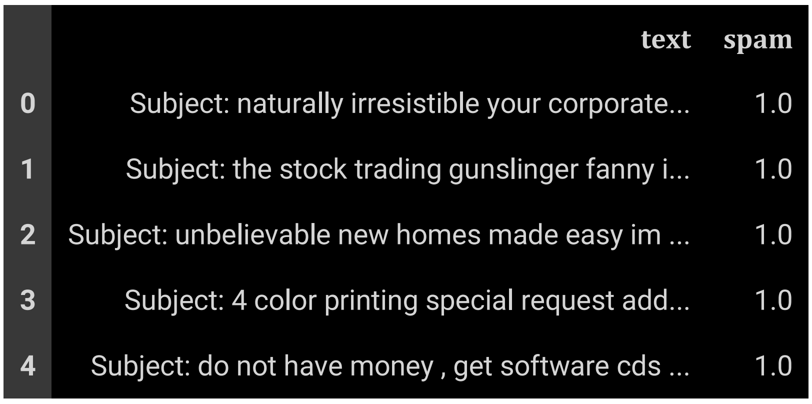
```
text     10862
spam     10862
dtype: int64
```

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
```

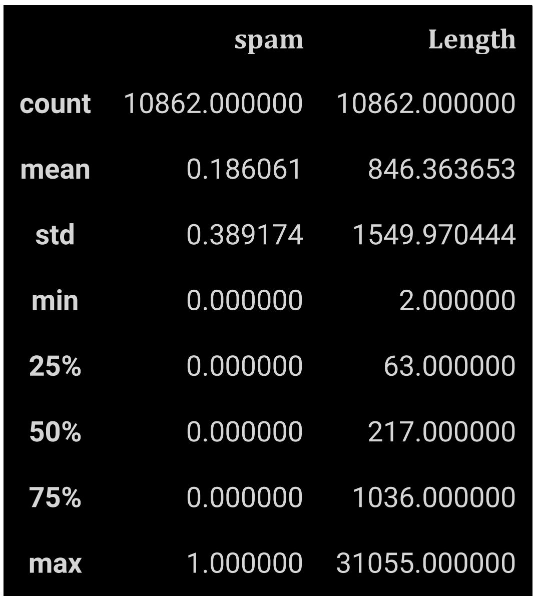
Text Preprocessing

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

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

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


data['Length'].max()
```

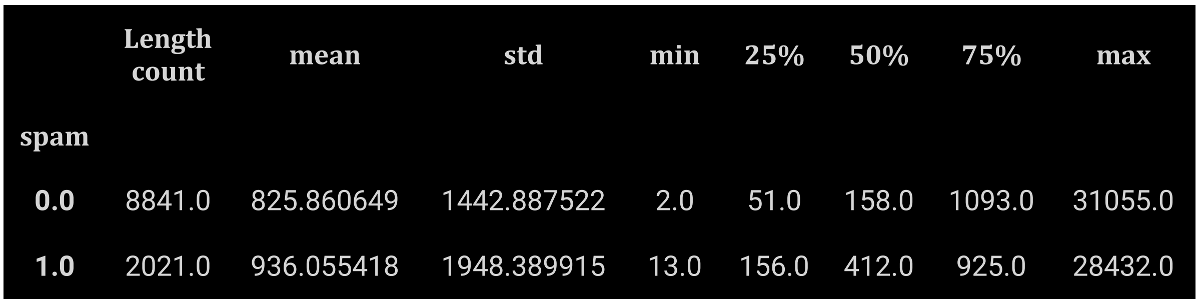
```
31055
```

```
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**

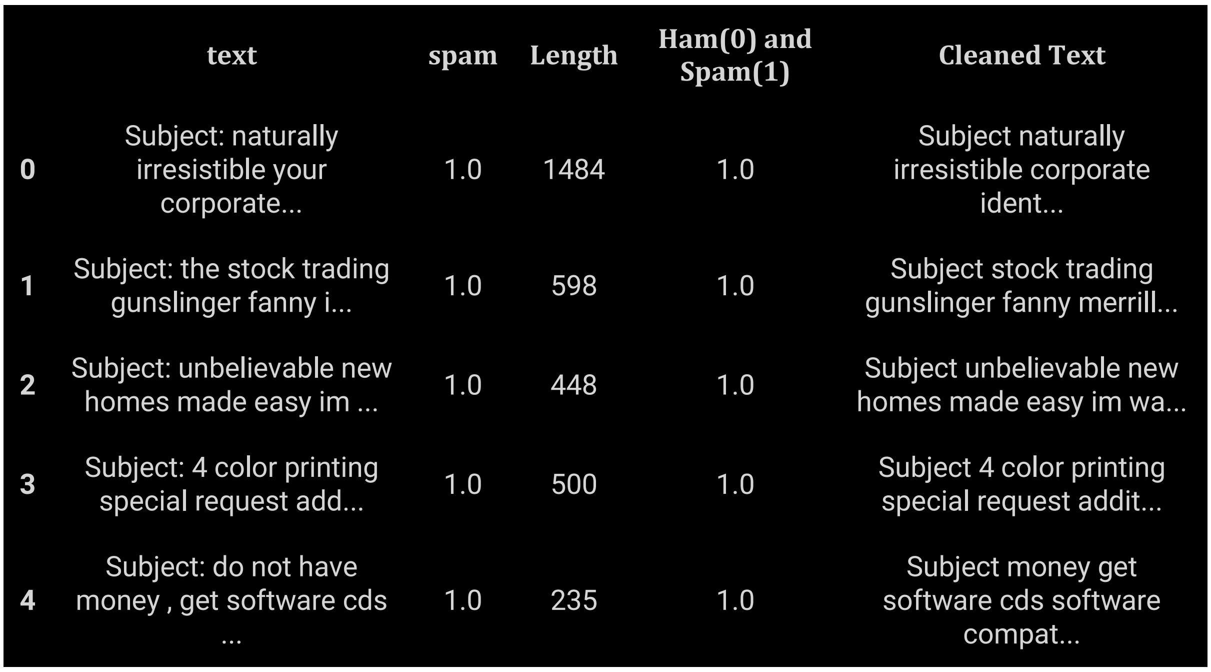
- Stop Words are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc).
- They don't add much information to the text.
- Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.

- 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 stopw
ords.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...

Porter Stemmer

- Tokenization
- Removing special characters
- Removing stop words and punctuation
- Stemming

```
from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

def transform_text(text):

    text = text.lower()
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()
    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        y.append(ps.stem(i))


    return " ".join(y)
data['Cleaned Text'] = data['Cleaned Text'].apply(transform_text)
data.head()
```

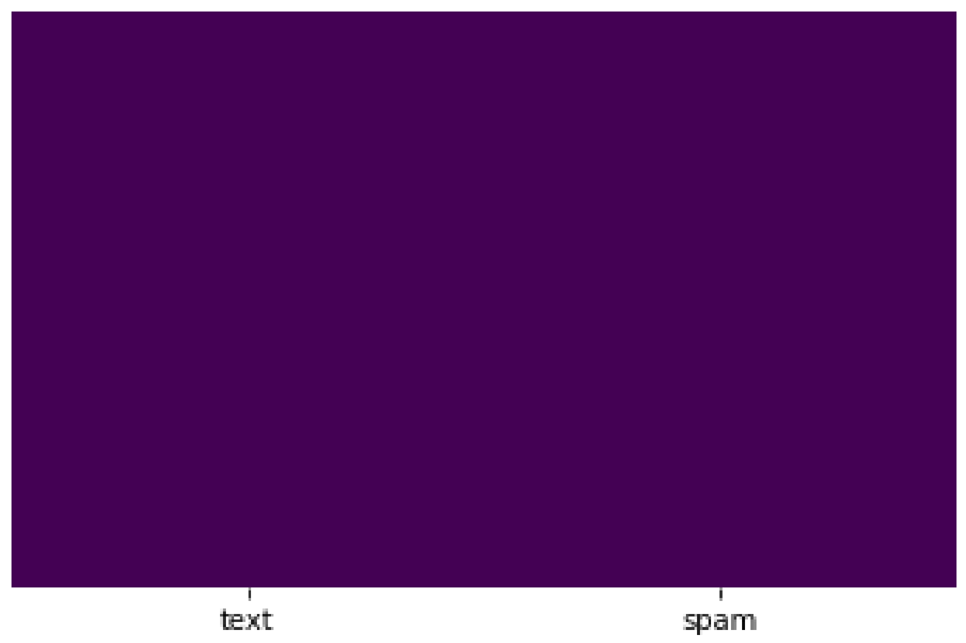
Data Visualization

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.

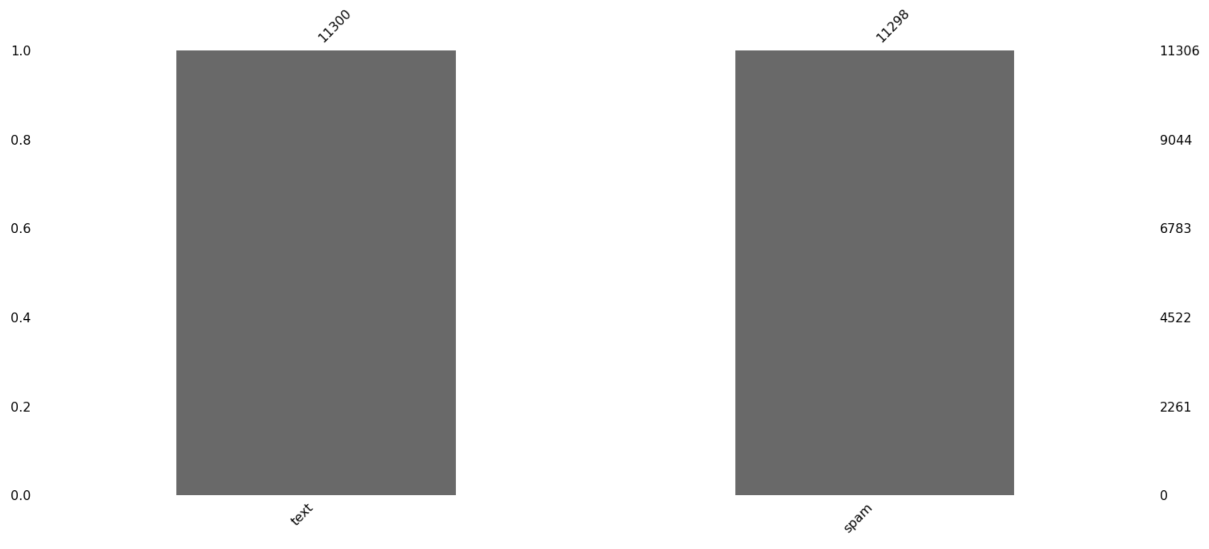
```
# To Check missing value
Import Seaborn
import seaborn as sn

# 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.

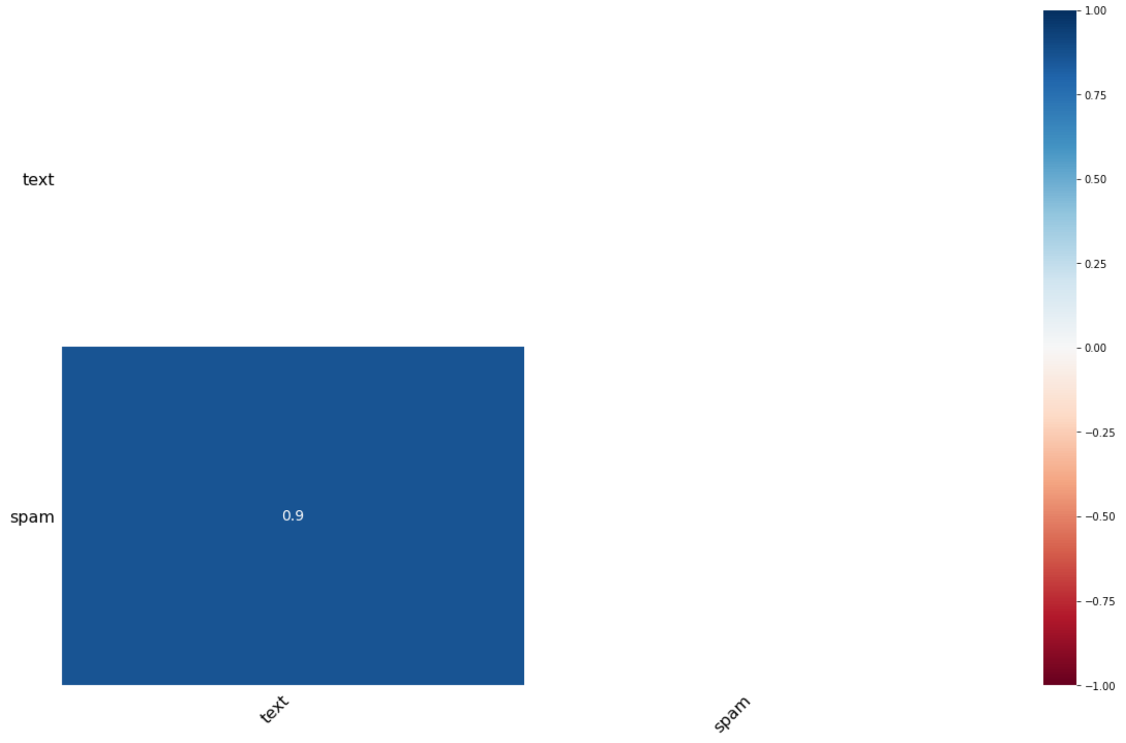
```
# Import missingno Library
import missingno as msno

# 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)

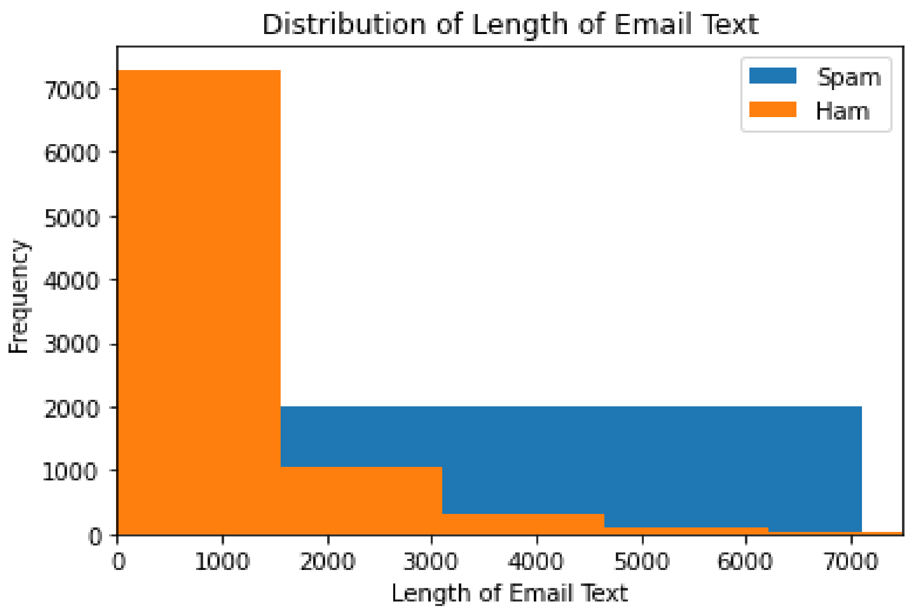


Pyplot the length of Spam & Ham Texts

- 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.

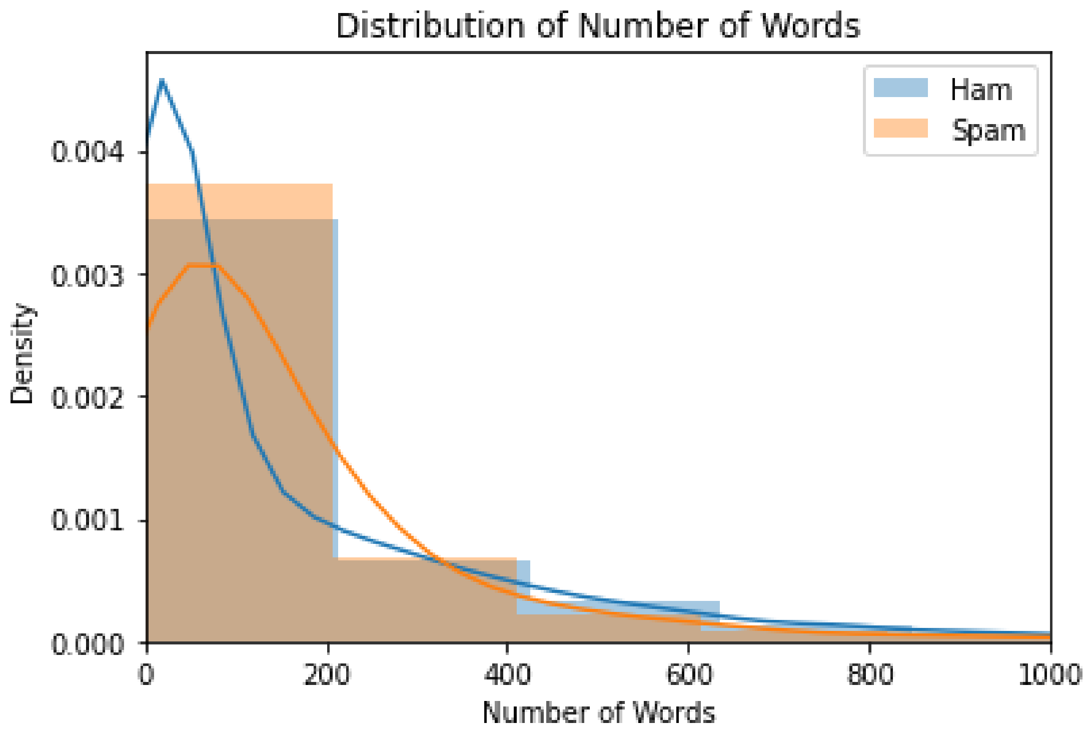
❖ One of the axes of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

```
# 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.xlim(0, 7500);
plt.legend()
```



Distplot the Spam & Ham record’s length after tokenizing

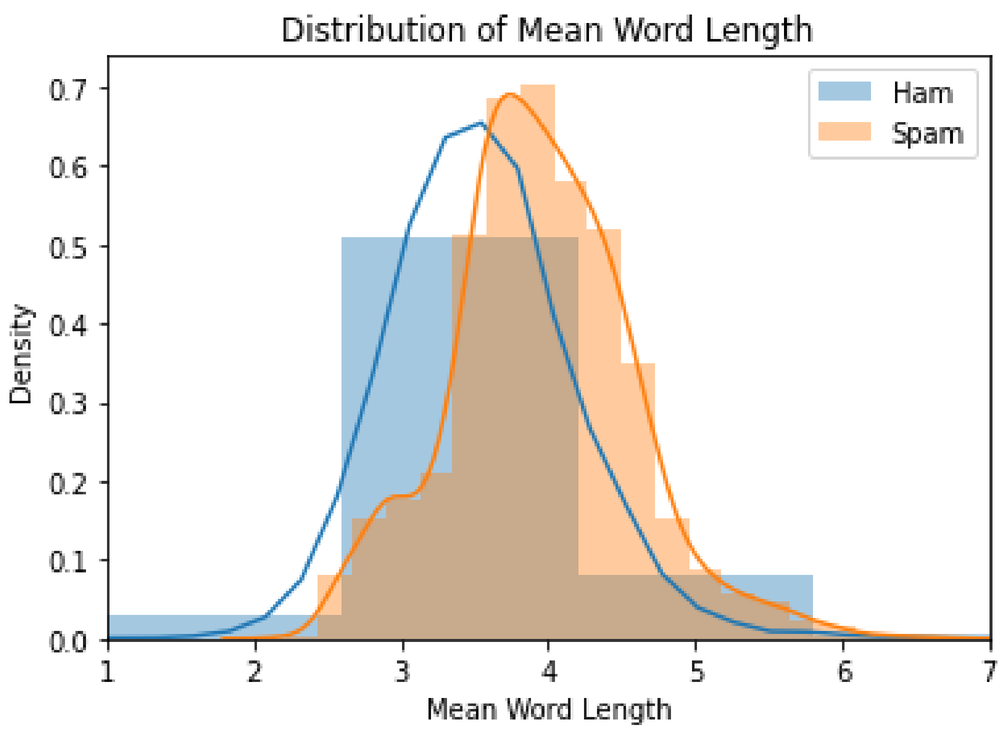
```
ax = sn.distplot(Ham_Words_Length, norm_hist = True, bins = 30, label = 'Ha
m')

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

print()

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

plt.xlabel('Number of Words')

plt.legend()

plt.xlim(0, 1000);

plt.show()
```



Distplot the Mean Word Length

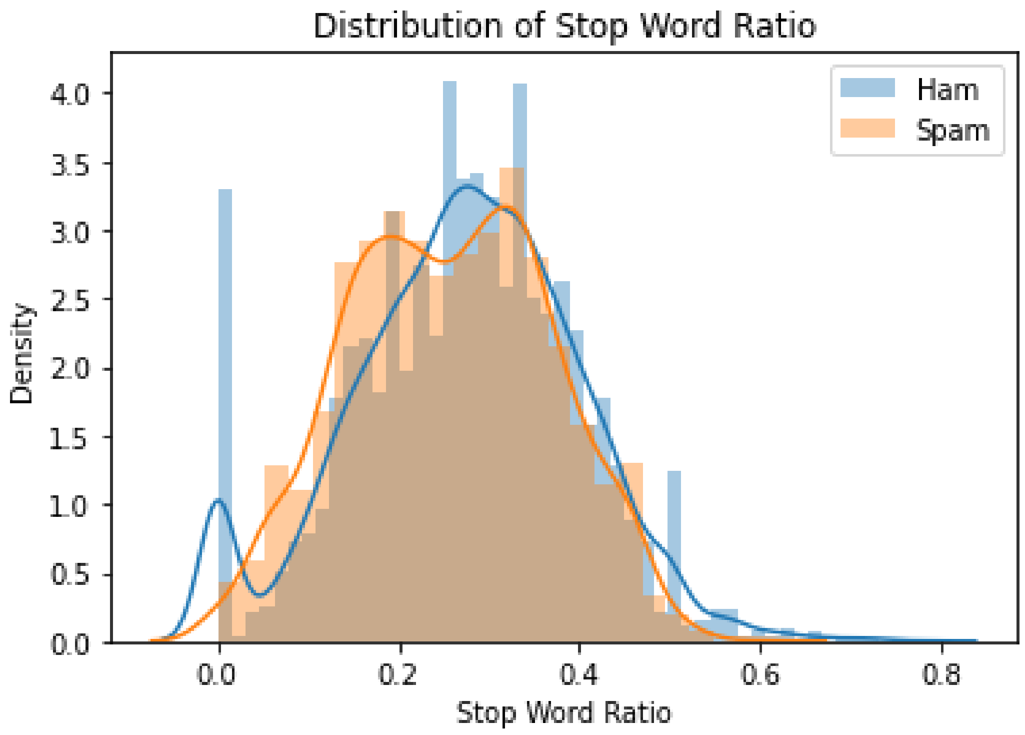
- A Distplot or distribution plot, depicts the variation in the data distribution.
- Seaborn Distplot represents the overall distribution of continuous data variables.
- The Seaborn module along with the Matplotlib module is used to depict the distplot with different variations in it.
- The Distplot 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 = 'Ha
m')

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

print()

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

plt.xlabel('Mean Word Length')

plt.legend()

plt.xlim(1, 7);
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()
```

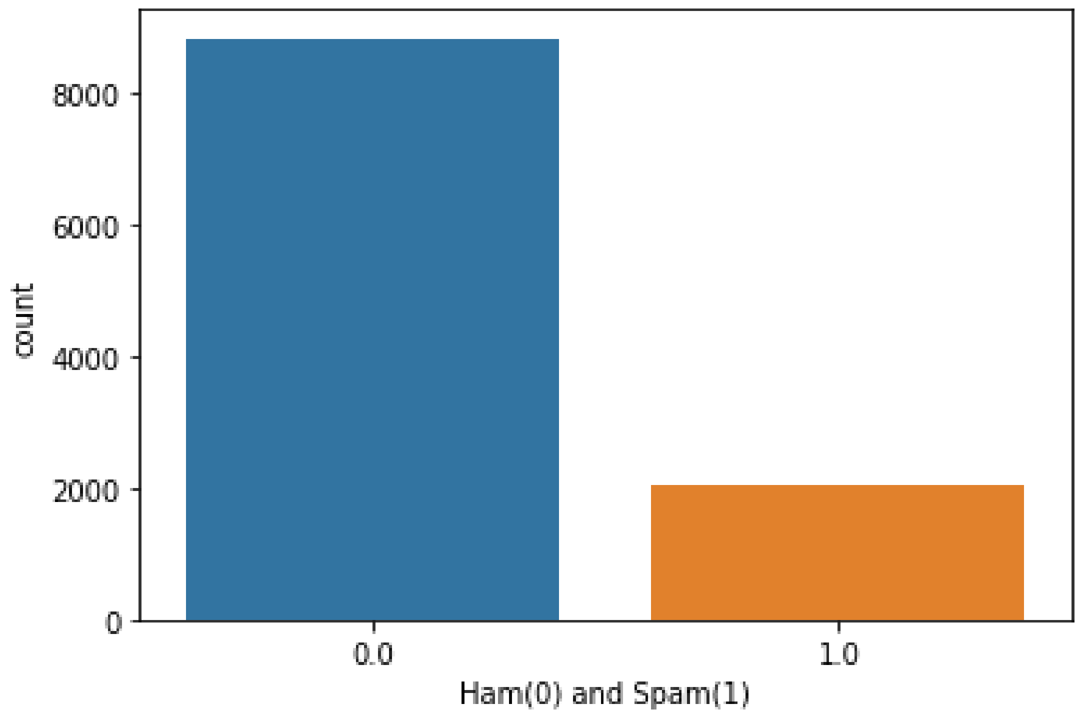


Countplot the Spam & Ham ratio

- ❖ 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 ra nge of values into a series of intervals—and then count how many val ues fall into each interval.
- ❖ The bins are usually specified as consecutive, non-overlapping interval s of a variable.
- ❖ The bins (intervals) must be adjacent and are often (but are not require d to be) of equal size.
- ❖ The x-axis of the histogram denotes the number of bins while the y-axi s represents the frequency of a particular bin.

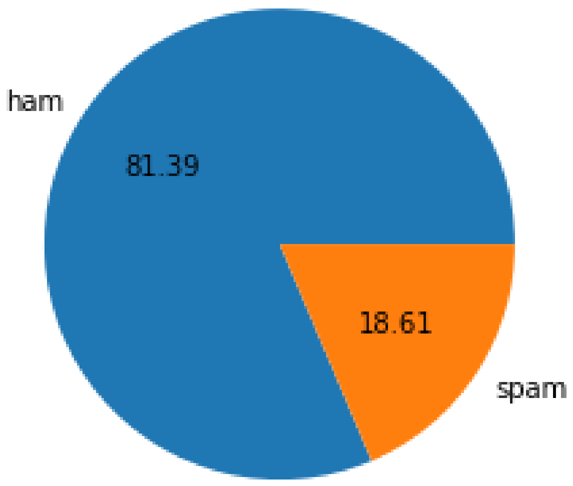
- The number of bins is a parameter which can be varied based on how you want to visualize the distribution of your data.

```
# 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")
```



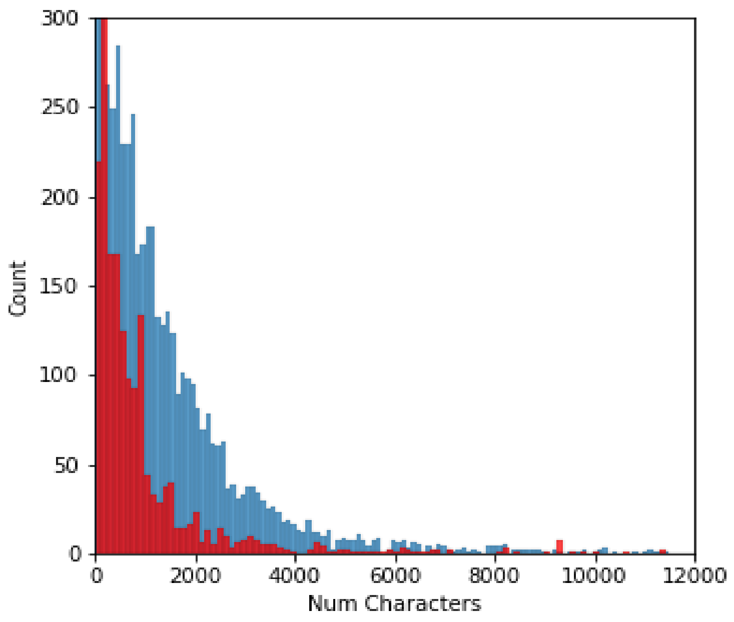
Pie Plot the Spam & Ham Ratio

```
import matplotlib.pyplot as plt

plt.pie(data['spam'].value_counts(), labels=['ham','spam'],autopct="%0.2f")

plt.show()
```

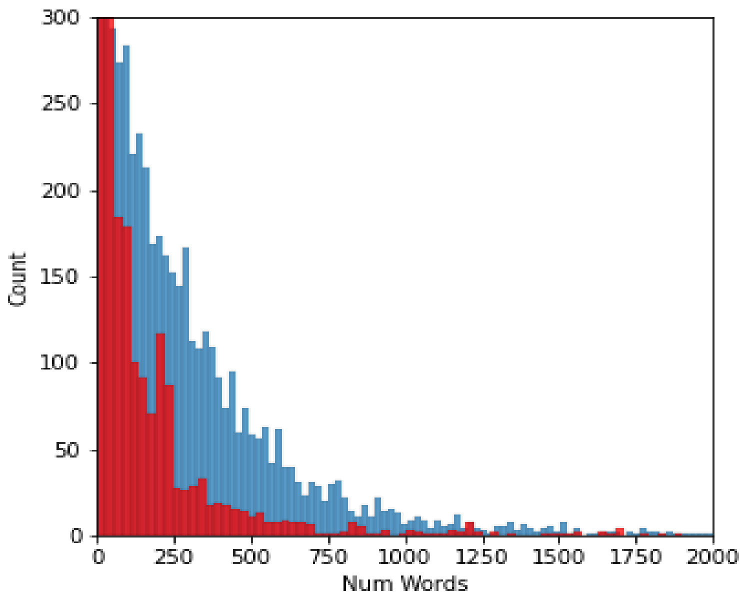


Histplot the number of characters, words and sentences

```
data['Num Characters'] = data['text'].apply(len)

data['Num Words'] = data['text'].apply(lambda x:len(nltk.word_tokenize(x)))

data['Num Sentences'] = data['text'].apply(lambda x:len(nltk.sent_tokenize(x))
)

import seaborn as sns

plt.figure(figsize=(5,5))

sns.histplot(data[data['spam'] == 0]['Num Characters'])

sns.histplot(data[data['spam'] == 1]['Num Characters'], color='red')

plt.xlim(0, 12000);

plt.ylim(0, 300);
```

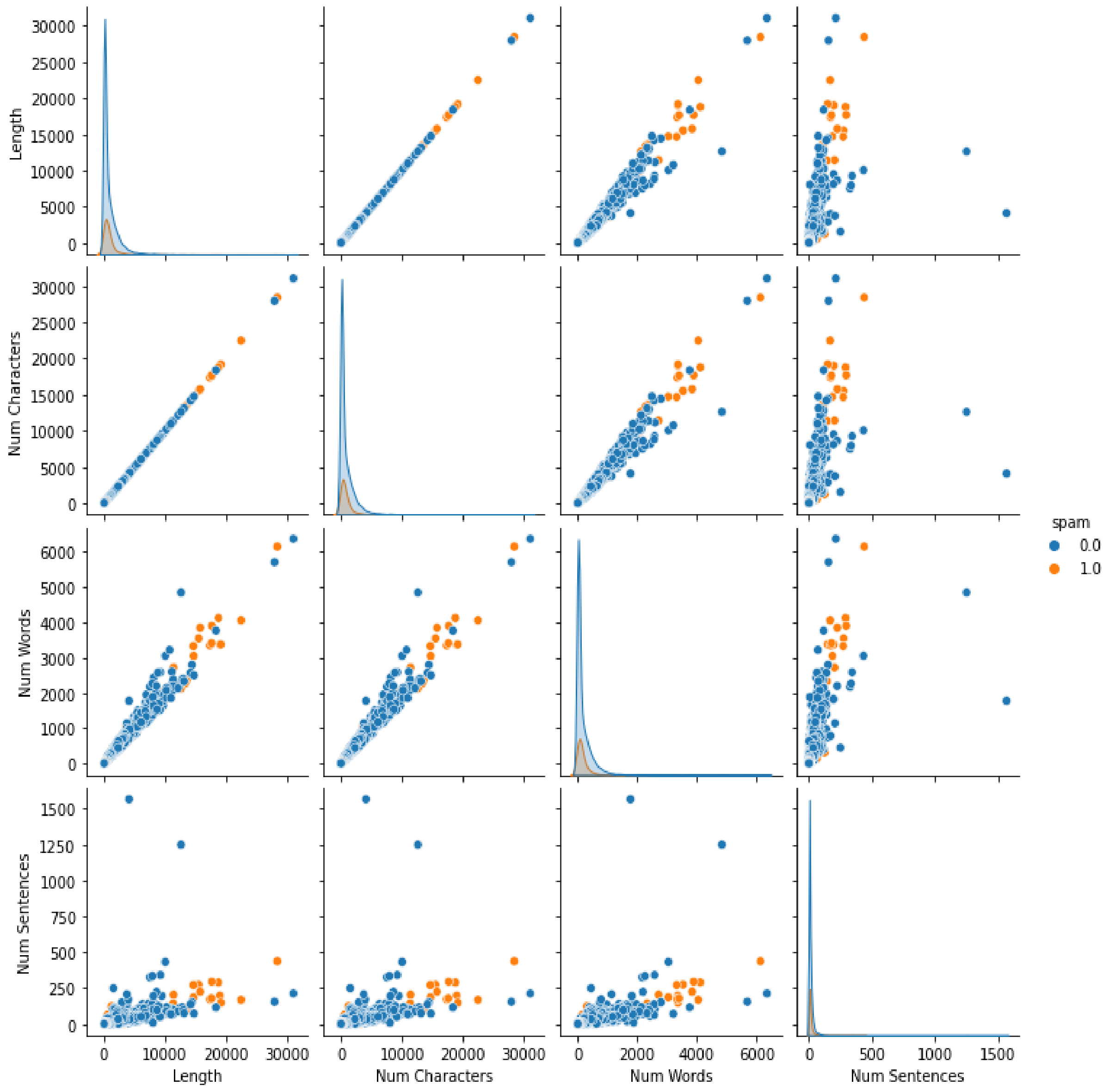


```
import seaborn as sns
plt.figure(figsize=(5,5))
sns.histplot(data[data['spam'] == 0]['Num Words'])
sns.histplot(data[data['spam'] == 1]['Num Words'], color='red')
plt.xlim(0, 2000);
plt.ylim(0, 300);
```



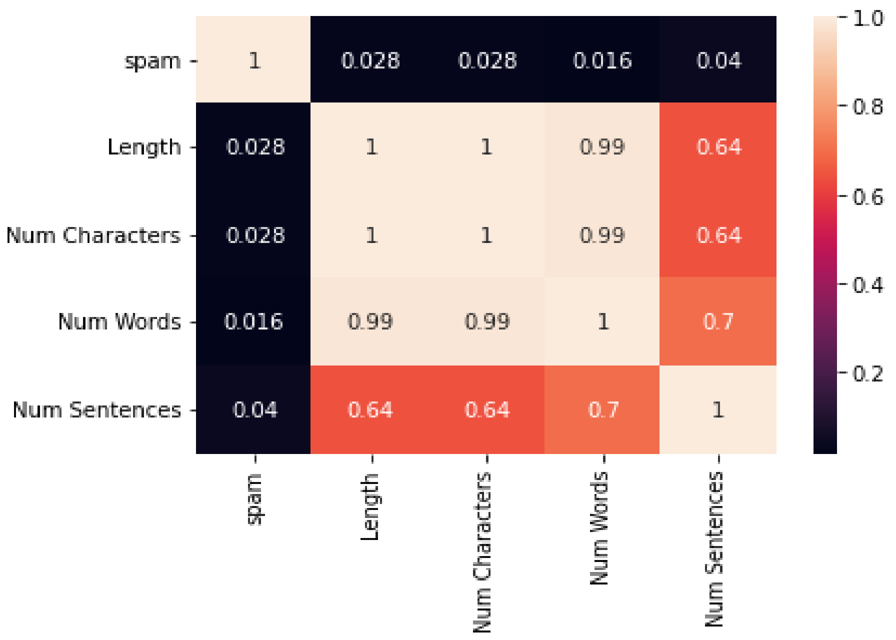
Pair plot the number of characters, words and sentences

```
sns.pairplot(data, hue='spam')
```



Plot the Heatmap of the dataset

```
sns.heatmap(data.corr(), annot=True)
```



	spam	Length	Num Characters	Num Words	Num Sentences
spam	1	0.028	0.028	0.016	0.04
Length	0.028	1	1	0.99	0.64
Num Characters	0.028	1	1	0.99	0.64
Num Words	0.016	0.99	0.99	1	0.7
Num Sentences	0.04	0.64	0.64	0.7	1

Word Cloud Visualization

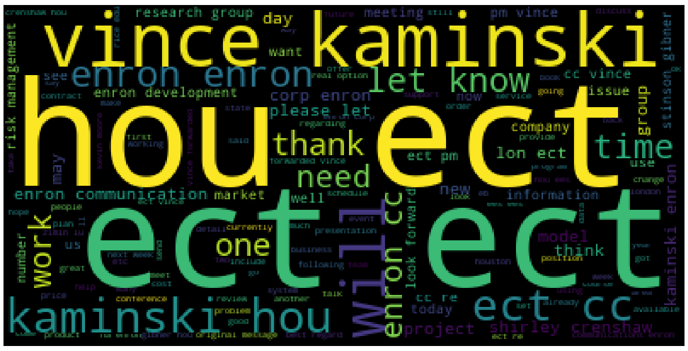
❖ Word cloud is a technique for visualising frequent words in a text whe re the size of the words represents their frequency.

❖ A word cloud (also called tag cloud or weighted list) is a visual represe ntation of text data. Words are usually single words, and the importan ce of each is shown with font size or color.

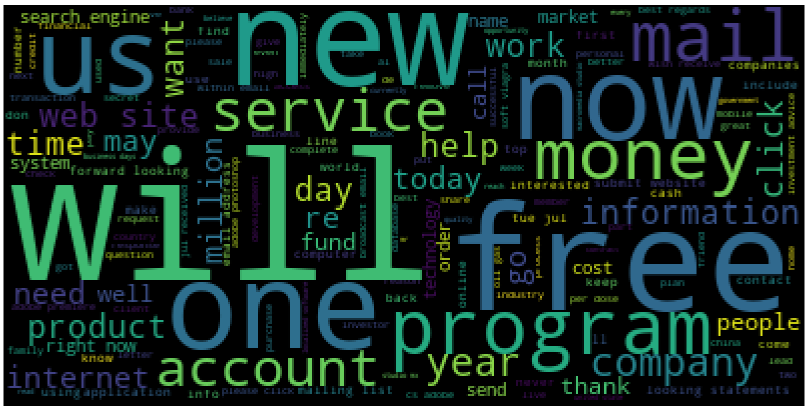
❖ Python fortunately has a wordcloud library allowing to build them.

❖ The wordcloud library is here to help you build a wordcloud in minute s 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=stopwo
rds, max_font_size = 90, margin=0, background_color = "black").generate(tex
t)

        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")
```


hou ect
ect ect
vince kaminski
kaminski hou
will
enron enron
let know
thank
time
need
one
ect cc
work

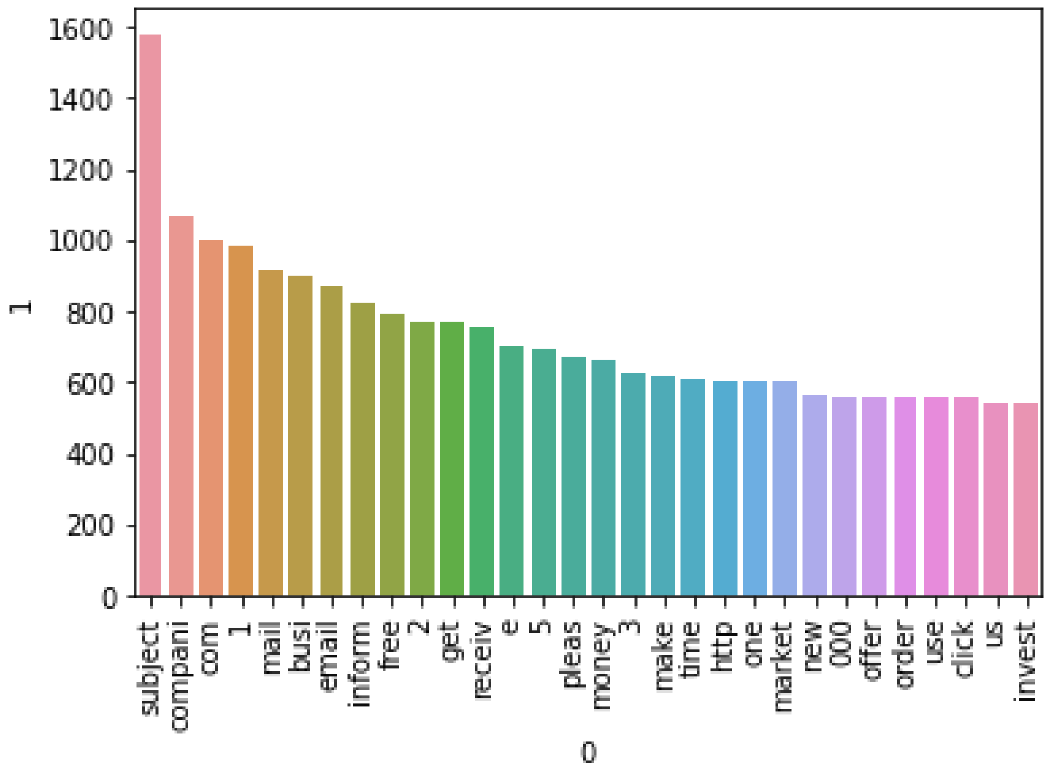
ect ect
vince kaminski
hou ect
enron enron
ect cc
will
kaminski hou
thank
let know
time
need
shirley crenshaw
enron communication


will
free
new
us
now
mail
one
program
money
service
account
internet
company
year
information
web site

will
free
new
now
one
us
mail
program
money
service
account
company
click
information
product

Spam Corpus

```
spam_corpus = []

for msg in data[data['spam'] == 1]['Cleaned Text'].tolist():

    for word in msg.split():

        spam_corpus.append(word)

len(spam_corpus)

185041

from collections import Counter

sns.barplot(pd.DataFrame(Counter(spam_corpus).most_common(30))[0],pd.DataFrame(Counter(spam_corpus).most_common(30))[1])

plt.xticks(rotation='vertical')

plt.show()
```

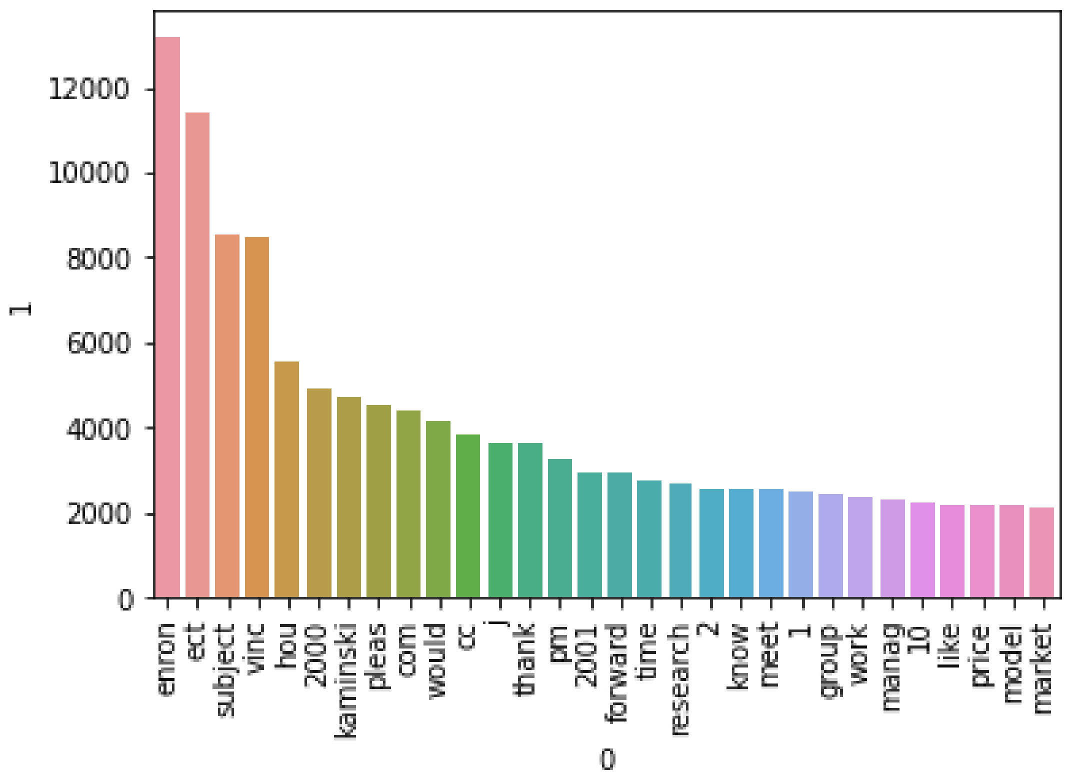


Ham Corpus

```
ham_corpus = []

for msg in data[data['spam'] == 0]['Cleaned Text'].tolist():

    for word in msg.split():

        ham_corpus.append(word)
len(ham_corpus)
```

```
731159
```

```
from collections import Counter

sns.barplot(pd.DataFrame(Counter(ham_corpus).most_common(30))[0],pd.DataFrame(Counter(ham_corpus).most_common(30))[1])

plt.xticks(rotation='vertical')

plt.show()
```



Classification Algorithms

KNN

What is KNN?

- KNN is a supervised machine learning algorithm whose goal is to learn a function such that f(X) = Y where X is the input, and Y is the output.
- KNN can be used both for classification as well as regression.
- It is non parametric as it does not make an assumption about the underlying data distribution pattern.
- Lazy algorithm as KNN does not have a training step. All data points will be used only at the time of prediction, and thus the prediction step is costly.
- KNN is an eager learner algorithm eagerly learns during the training step.
- KNN uses feature similarity to predict the cluster that the new point will fall into.

Working of KNN

- In the training phase, the model will store the data points.
- In the testing phase, the distance from the query point to the points from the training phase is calculated to classify each point in the test dataset.
- Various distances can be calculated, but the most popular one is the Euclidean distance (for smaller dimension data).

- Other distance measures such as Manhattan, Hamming, and Chebyshev distance can also be used based on the data.

For Example: We have 500 N-dimensional points, with 300 being class 0 and 200 being class 1. The procedure for calculating the class of query point is:

- ✓ The distance of all the 500 points is calculated from the query point.
- ✓ Based on the value of K, K nearest neighbours are used for the comparison purpose.
- ✓ Let's say K=7, 4 out of 7 points are of class 0, and 3 are of class 1. Then based on the majority, the query point p is assigned as class 0.

What happens when K changes?

- K=1 means that it will take one nearest neighbour and classify the query point based on that. The surface that divides the classes will be very uneven (many vertices).
- The problem that arises here is if an outlier is present in the data, the decision surface considers that as a data point.
- Due to this, KNN will perform exceptionally well on the training dataset but will misclassify many points on the test dataset (unseen data).
- This is considered as overfitting, and therefore, KNN is sensitive to outliers.

- As the value of K increases, the surface becomes smooth and will not consider the outliers as data points. This will better generalize the model on the test dataset also.
- If K value is extremely large, the model will underfit and will be unable to classify the new data point.

For example: If K is equal to the total number of data points, no matter where the query point lies, the model will always classify the query point based on the majority class of the whole dataset.

How to select appropriate K?

- In real-world problems, the dataset is separated into three parts, namely, training, validation, and test data.
- In KNN, the training data points get stored, and no learning is performed.
- Validation data is to check the model performance, and the test data is used for prediction.
- To select optimal K, plot the error of model (error = 1 — accuracy) on training as well as on the validation dataset.
- The best K is where the validation error is lowest, and both training and validation errors are close to each other.

Limitation of KNN

- Time complexity and space complexity is enormous, which is a major disadvantage of KNN.
- Time complexity refers to the time model takes to evaluate the class of the query point and the space complexity refers to the total memory used by the algorithm.
- If we have n data points in training and each point is of m dimension, then time complexity is of order O(nm), which will be huge if we have higher dimension data and hence KNN is not suitable for high dimensional data.
- Another disadvantage is if the data point is far away from the classes present (no similarity), KNN will classify the point even if it is an outlier.
- In order to overcome the problem of time complexity, algorithms such as KD-Tree and Locality Sensitive Hashing (LSH) can be used.

KNN Implementation from Scratch

Model Building

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

X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

Y = data['spam'].values
```

Splitting the Dataset

```
from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split( X, Y, test_size = 0.2, random_
state = 2 )

X_train, X_test, Y_train, Y_test = train_test_split( X_train, Y_train, test_size =
0.2, random_state = 2 )
```

Standardization

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)

X_val= scaler.transform(X_val)
```

Finding the best hyper parameter (K)

```
from sklearn.neighbors import KNeighborsClassifier

import sklearn.metrics as metrics

for i in range(1, 11):

  classifier = KNeighborsClassifier(n_neighbors = i)

  classifier.fit(X_train, Y_train)

  Y_pred = classifier.predict(X_val)

  print("Accuracy for k :", i,"is", metrics.accuracy_score(Y_val, Y_pred))
```

```
Accuracy for k : 1 is 0.9406350667280258

Accuracy for k : 2 is 0.9438564196962724

Accuracy for k : 3 is 0.9415554532903819

Accuracy for k : 4 is 0.9475379659456972

Accuracy for k : 5 is 0.9498389323515877

Accuracy for k : 6 is 0.9507593189139438

Accuracy for k : 7 is 0.9521398987574782

Accuracy for k : 8 is 0.9516797054763001

Accuracy for k : 9 is 0.9544408651633686

Accuracy for k : 10 is 0.9535204786010124
```

- Its evident that when **K = 1** the accuracy is high**.** So let’s take the value of k as 1.

Training the Model on the train data

```
from scipy.stats import mode

class KNN() :

  def __init__( self, K ) :

    self.K = K


  def fit( self, X_train, Y_train ) : # Function to store training set

    self.X_train, self.Y_train = X_train, Y_train

    self.m, self.n = X_train.shape # No of Rows & Columns in Training Data Set


  def predict( self, X_test ) : # Function for prediction

    self.X_test = X_test

    self.m_test, self.n = X_test.shape # No of Rows & Columns in Test Data Set

    Y_predict = np.zeros( self.m_test )

    for i in range( self.m_test ) :

      neighbors = self.find_neighbors( self.X_test[i] ) # Find the K nearest neigh
bors from current test example

      # most frequent class in K neighbors

      Y_predict[i] = mode( neighbors )[0][0]

    return Y_predict


  def find_neighbors( self, x ) : # Function to find the K nearest neighbors to c
urrent test example

    # Calculate all the Euclidean distances between current test example x and
training set X_train
```

```
    euclidean_distances = np.zeros( self.m )

    for i in range( self.m ) :

      euclidean_distances[i] = self.euclidean( x, self.X_train[i] )

      Y_train_sorted = self.Y_train[euclidean_distances.argsort()] # Sort Y_train
according to euclidean_distance_array and store it in Y_train_sorted

    return Y_train_sorted[:self.K]


  def euclidean( self, x, x_train ) : # Function to calculate euclidean distance

    return np.sqrt( np.sum( np.square( x - x_train ) ) )


model = KNN( K = 1 )

model.fit( X_train, Y_train )

classifier.fit(X_train, Y_train)
```

Prediction on test set

```
Y_pred = classifier.predict(X_test)
```

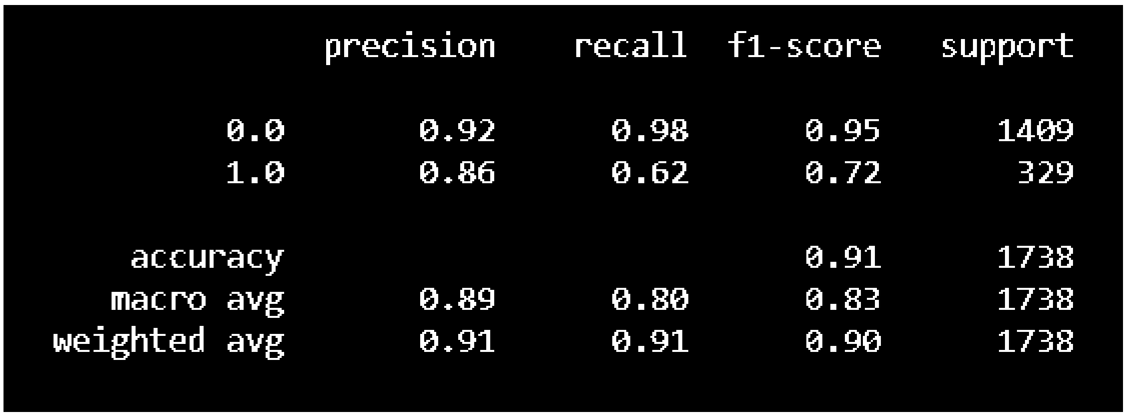
Accuracy

```
import sklearn.metrics as metrics

from sklearn.metrics import precision_score, \
    recall_score, confusion_matrix, classification_report, \
    accuracy_score, f1_score

print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

print("Precision:",metrics.precision_score(Y_test, Y_pred))

print("Recall:",metrics.recall_score(Y_test, Y_pred))
print("F1 Score :",f1_score(Y_test,Y_pred))
```

```
Accuracy: 0.9096662830840047

Precision: 0.864406779661017

Recall: 0.6200607902735562

F1 Score : 0.7221238938053097
```

Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(Y_test, Y_pred))
```



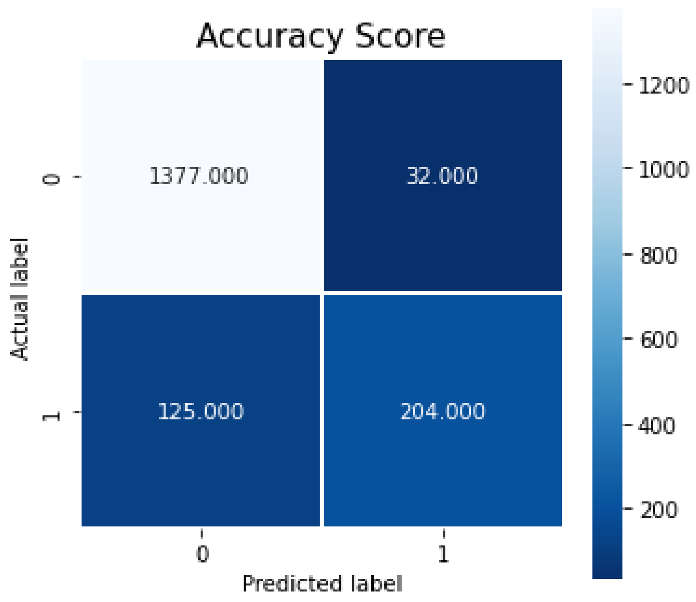
```
              precision    recall  f1-score   support

         0.0       0.92      0.98      0.95      1409
         1.0       0.86      0.62      0.72       329

    accuracy                           0.91      1738
   macro avg       0.89      0.80      0.83      1738
weighted avg       0.91      0.91      0.90      1738
```

Confusion Matrix

```
plt.figure(figsize=(5,5))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap
= 'Blues_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');
```

all_sample_title = 'Accuracy Score'

plt.title(all_sample_title, size = 15);



KNN Using Sklearn

Model Building

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors = 1)

classifier.fit(X_train, Y_train)

Prediction on test set

Y_pred = classifier.predict(X_test)

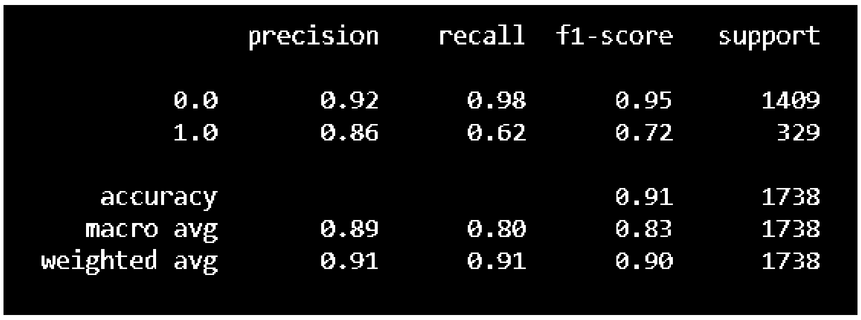
Accuracy

```
import sklearn.metrics as metrics

from sklearn.metrics import precision_score,\ recall_score, confusion_matrix
, classification_report, \  accuracy_score, f1_score

print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

print("Precision:",metrics.precision_score(Y_test, Y_pred))

print("Recall:",metrics.recall_score(Y_test, Y_pred))
print("F1 Score :",f1_score(Y_test,Y_pred))
```

```
Accuracy: 0.9096662830840047

Precision: 0.864406779661017

Recall: 0.6200607902735562

F1 Score : 0.7221238938053097
```

Classification Report

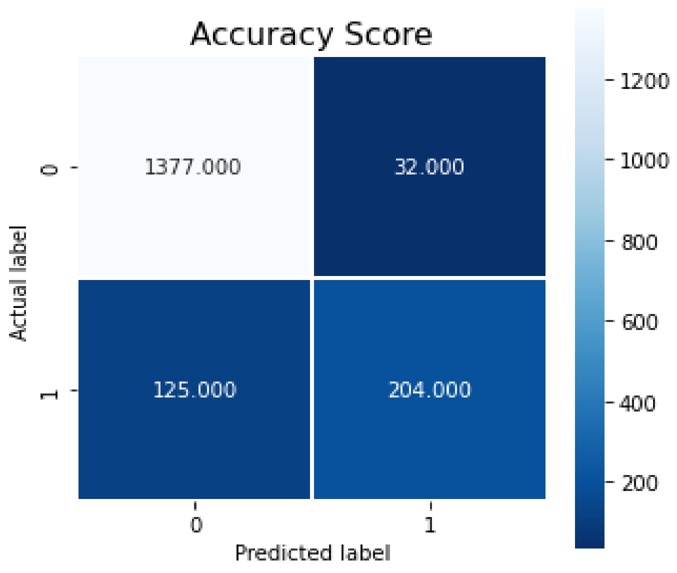
```
from sklearn.metrics import classification_report

print(classification_report(Y_test, Y_pred))
```



```
              precision    recall  f1-score   support

         0.0       0.92      0.98      0.95      1409
         1.0       0.86      0.62      0.72       329

    accuracy                           0.91      1738
   macro avg       0.89      0.80      0.83      1738
weighted avg       0.91      0.91      0.90      1738
```

Confusion Matrix

```
plt.figure(figsize=(5,5))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap
= 'Blues_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');

all_sample_title = 'Accuracy Score'

plt.title(all_sample_title, size = 15);
```



Validation Of KNN

Simple K Fold

Simple K Fold using Library

```
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator = classifier, X = X_train, y = Y_train, c
v = 10)

print("Accuracy: {:.2f} %".format(accuracies.mean()*100))

accuracies
```

Simple K Fold from scratch

```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

cv = CountVectorizer()

tfidf = TfidfVectorizer(max_features=3000)

X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

Y = data['spam'].values

from sklearn.model_selection import KFold

accuracy1 = []

kf = KFold(n_splits=5, random_state=None)

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    # Predicting Test Data Set
    Y_pred = model.predict( X_test )

    # Confusion Matrix
    print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end =
"\n")

    # F1 Score
    print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
    accuracy1.append(accuracy_score(Y_test, Y_pred))
    print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

```
Accuracy: 90.85 %

array([0.89798851, 0.91510791, 0.89640288, 0.92517986, 0.90503597,
       0.91366906, 0.90791367, 0.90935252, 0.91079137, 0.90359712])
```

Stratified K Fold

Stratified K Fold using Library

```
from sklearn.model_selection import StratifiedKFold

from sklearn.model_selection import cross_val_score

skf = StratifiedKFold(n_splits=5, random_state=None)

accuracies = cross_val_score(estimator = classifier, X = X_train, y = Y_train, c
v = skf)

print("Accuracy: {:.2f} %".format(accuracies.mean()*100))

accuracies
```

Stratified K Fold from scratch

```
from sklearn.model_selection import StratifiedKFold

accuracy2 = []

skf = StratifiedKFold(n_splits=5, random_state=None)

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )
    # Predicting Test Data Set
    Y_pred = model.predict( X_test )

    # Confusion Matrix
    print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end =
"\n")

    # F1 Score
    print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
    accuracy2.append(accuracy_score(Y_test, Y_pred))
    print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

```
Accuracy: 90.29 %

array([0.89863408, 0.9057554 , 0.90863309, 0.90071942, 0.90071942])
```

Repeated Random Split

```
from sklearn.model_selection import RepeatedKFold
accuracy3 = []
kf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)
for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )
    # Predicting Test Data Set
    Y_pred = model.predict( X_test )

    # Confusion Matrix
    print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end =
"\n")

    # F1 Score
    print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
    accuracy3.append(accuracy_score(Y_test, Y_pred))
    print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

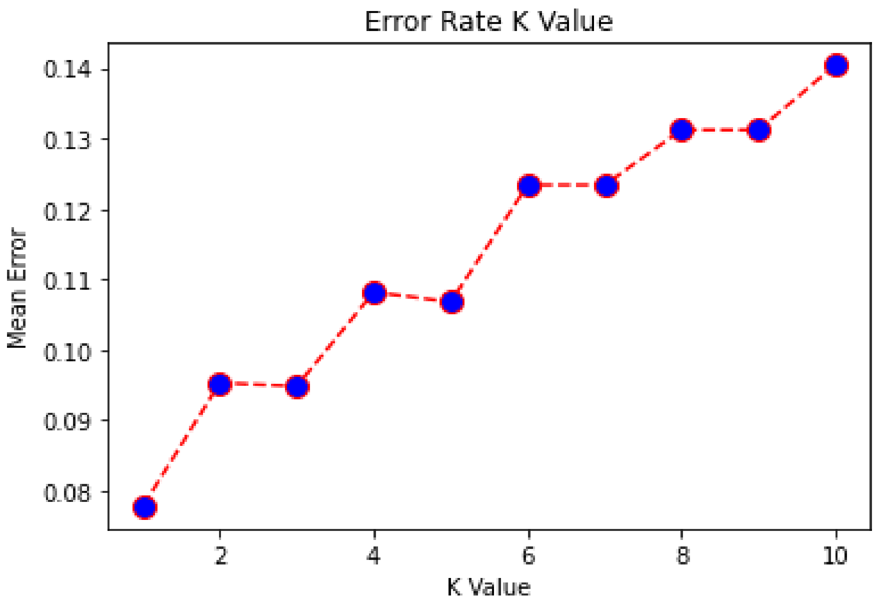
Accuracy Check

```
print("Mean Accuracy of K Fold : ", np.mean(accuracy1))

print("Mean Accuracy of Stratified K Fold : ", np.mean(accuracy2))

print("Mean Accuracy of Repeated K Fold : ", np.mean(accuracy3))
```

```
Mean Accuracy of K Fold :  0.8344949188051247

Mean Accuracy of Stratified K Fold :  0.8344949188051247

Mean Accuracy of Repeated K Fold :  0.8859695331707824
```

Various K Comparisons

```
from sklearn.neighbors import KNeighborsClassifier

error = []


for i in range(1, 11):  # Calculating error for K values between 1 and 10

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, Y_train)

    pred_i = knn.predict(X_test)

    error.append(np.mean(pred_i != Y_test))
plt.figure(figsize=(6, 4));

plt.plot(range(1,11), error, color='red', linestyle='dashed', marker='o',
markerfacecolor='blue', markersize=10);

plt.title('Error Rate K Value');

plt.xlabel('K Value');

plt.ylabel('Mean Error');
```



- It’s evident that when the **K** value is 1, the mean error rate is very less and it increases as the value of **K** increase

Logistic Regression

What is Logistic Regression?

- It’s a classification algorithm, that is used where the response variable is categorical.
- The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.
- For example: When we have to predict if a student passes or fails in an exam when the number of hours spent studying is given as a feature, the response variable has two values, pass and fail.
- This type of a problem is referred to as Binomial Logistic Regression, where the response variable has two values 0 and 1 or pass and fail or true and false.
- Multinomial Logistic Regression deals with situations where the response variable can have three or more possible values.

Why Logistic, not Linear?

- With binary classification, let ‘x’ be some feature and ‘y’ be the output which can be either 0 or 1.

Types of Logistic Regression

- Binary Logistic Regression
    - The dependent variable has only two 2 possible outcomes/classes.
    - Example-Male or Female.
- Multinomial Logistic Regression
    - The dependent variable has only two 3 or more possible outcomes/classes without ordering.
    - Example: Predicting food quality.(Good,Great and Bad).
- Ordinal Logistic Regression
    - The dependent variable has only two 3 or more possible outcomes/classes with ordering.
    - Example: Star rating from 1 to 5

Assumptions of Logistic Regression

Even though Logistic Regression belongs to the Linear models, it does not make any assumptions of the Linear Regression models, like,

- The error terms do not need to be normally distributed.
- Homoscedasticity is not required.

However, it has few of its own assumption:

- It assumes that there is minimal, or no multi-collinearity among the independent variables.
- It assumes that independent variables that linearly related to log of odds.
- It assumes a large sample for good prediction.
- It assumes that the observations are independent of each other.

Logistic Regression with 2 classes that the dependent variable is binary and the ordered Logistic Regression requires the dependent variable to be ordered.

The Logistic Model

- The Logistic Regression instead for fitting the best fit line, condenses the output of the linear function between 0 and 1.

Interpretation of the Coefficients

- Interpretation of the weights differ from the Linear Regression as the output of the Logistic Regression is in probabilities between 0 and 1.
- Instead of the slope co-efficient(b) being the rate of change of the p as x changes, now the slope co-efficient is interpreted as the rate of change of the "log odds" as X changes.

Odds Ratio and Logit

- Odds ratio is defined as the ratio of the odds in presence of B and odds of A in the absence of B and vice versa.

Decision Boundary

- A decision Boundary is a line or margin that separates the classes.
- Logistic Regression decides a proper fit to the decision boundary so that we will be able to predict which class a new data will correspond to.

Cost Function of the Logistic Regression

- Cost Function is a function that measures the performance of a Machine Learning model for given data.

- Cost Function is basically the calculation of the error between predicted values and expected values and presents it in the form of a single real number.

Gradient Descent in Logistic Regression

- Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function that minimizes a cost function (cost).
- The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a cost function.

Logistic Regression from Scratch

Model Building

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

X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

Y = data['spam'].values
```

Splitting the Dataset

```
from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split( X, Y, test_size = 0.2, random_
state = 2 )

X_train, X_test, Y_train, Y_test = train_test_split( X_train, Y_train, test_size =
0.2, random_state = 2 )
```

Standardization

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler(); scaler.fit(X_train)

X_train = scaler.transform(X_train); X_test = scaler.transform(X_test)

X_val= scaler.transform(X_val)
```

Finding the best solver

```
from sklearn.model_selection import GridSearchCV

param = {

    'solver' : ['newton-cg', 'lbfgs','saga','sag']
}

logreg = LogisticRegression()

grid_search = GridSearchCV(estimator = logreg, param_grid = param, cv = 5)

grid_result = grid_search.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result.best_score_, rid_result.best_params_))
```

```
Best: 0.960525 using {'solver': 'newton-cg'}
```

- So the best solver is the newton-cg.

Training the train data

```
class LogisticRegression:

    def __init__(self, learning_rate=0.01, n_iters=1000):

        """
        LogisticRegression constructor function and initial values.
        """

        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):

        """
        It ensures that weights are updated according to the incoming sample
and class information.
        """

        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0

        # Gradient Descent

        for _ in range(self.n_iters):

            # Approximate output variable (y) with linear combination of weights
and x, plus bias

            linear_model = np.dot(X, self.weights) + self.bias

            # Apply Sigmoid Function

            y_predicted = self._sigmoid(linear_model)
            # Compute Gradients

            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) # Derivative w.r.t
```

```
weights
            db = (1 / n_samples) * np.sum(y_predicted - y)  # Derivative w.r.t bias

            # Parameters Updation

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):

        """
        It enables the class prediction of a new sample sent with a parameter by
a pre-trained logistic regression model.
        """

        linear_model = np.dot(X, self.weights) + self.bias

        y_predicted = self._sigmoid(linear_model)

        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]

        return np.array(y_predicted_cls)

    def _sigmoid(self, x):

        """
        The Sigmoid function is used to equal the submitted numeric values to
a value in the range (0-1).
        """

        return 1 / (1 + np.exp(-x))


lr = LogisticRegression()

lr.fit(X_train, Y_train)
```

Prediction on the test data

```
Y_pred = lr.predict(X_test)
```

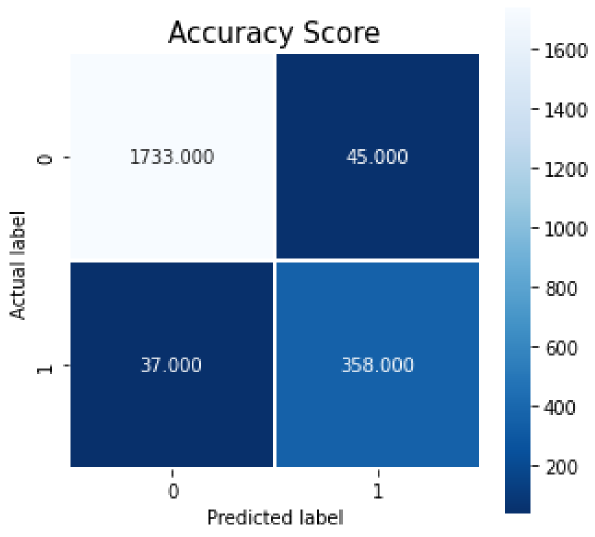
Accuracy

```
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

print("Precision:",metrics.precision_score(Y_test, Y_pred))

print("Recall:",metrics.recall_score(Y_test, Y_pred))
```

```
Accuracy: 0.9622641509433962

Precision: 0.8883374689826302

Recall: 0.9063291139240506
```

Confusion Matrix

```
plt.figure(figsize=(5,5))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');

plt.ylabel('Actual label'); plt.xlabel('Predicted label');

all_sample_title = 'Accuracy Score'; plt.title(all_sample_title, size = 15);
```



Logistic Regression using Sklearn

Training the Model

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(solver = 'newton-cg')

logreg.fit(X_train, Y_train)
```

Prediction the test data

```
Y_pred = logreg.predict(X_test)
```

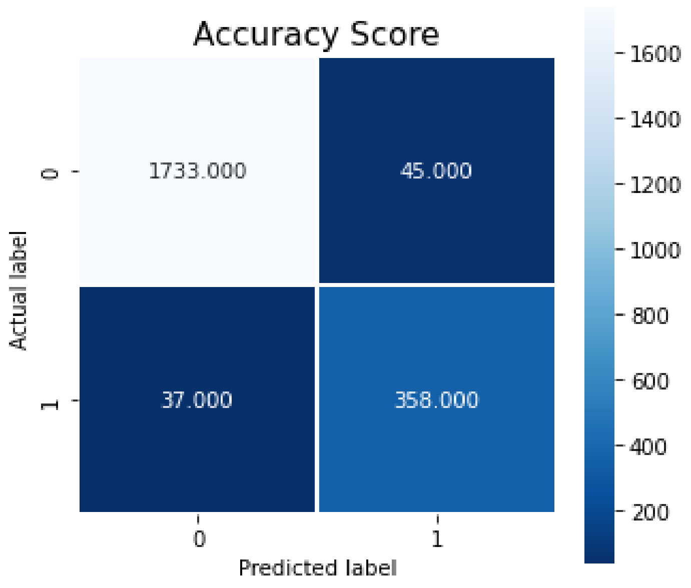
Accuracy

```
import sklearn.metrics as metrics

print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

print("Precision:",metrics.precision_score(Y_test, Y_pred))

print("Recall:",metrics.recall_score(Y_test, Y_pred))
```

```
Accuracy: 0.9622641509433962

Precision: 0.8883374689826302

Recall: 0.9063291139240506
```

Confusion Matrix

```
plt.figure(figsize=(5,5))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap
= 'Blues_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');

all_sample_title = 'Accuracy Score'

plt.title(all_sample_title, size = 15);
```



Validation of Logistic Regression

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import LeavePOut
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import StratifiedKFold
```

Simple K-fold Cross Validation

```
kfold = model_selection.KFold(n_splits=10, random_state=100)

model_kfold = LogisticRegression()

results_kfold = model_selection.cross_val_score(model_kfold, X, Y, cv=kfold)

print("Accuracy: %.2f%%" % (results_kfold.mean()*100.0))
```

```
Accuracy: 92.56%
```

Stratified K-fold Cross Validation

```
skfold = StratifiedKFold(n_splits=10, random_state=100)

model_skfold = LogisticRegression()

results_skfold = model_selection.cross_val_score(model_skfold, X, Y,
cv=skfold)

print("Accuracy: %.2f%%" % (results_skfold.mean()*100.0))
```

```
Accuracy: 96.34%
```

Leave One Out Cross-Validation (LOOCV)

```
loocv = model_selection.LeaveOneOut()

model_loocv = LogisticRegression()

results_loocv = model_selection.cross_val_score(model_loocv, X, Y, cv=loocv)

print("Accuracy: %.2f%%" % (results_loocv.mean()*100.0))
```

Repeated Random Test Train Splits

```
kfold2 = model_selection.ShuffleSplit(n_splits=10, test_size=0.20,
random_state=100)

model_shufflecv = LogisticRegression()

results_4 = model_selection.cross_val_score(model_shufflecv, X, Y,
cv=kfold2)

print("Accuracy: %.2f%% (%.2f%%)" % (results_4.mean()*100.0,
results_4.std()*100.0))
```

```
Accuracy: 96.68% (0.27%)
```

- It’s evident that the Repeated Random Test Train Split has higher accuracy.

SVM

- Support Vector Machines are supervised learning models for classification and regression problems. They can solve linear and non-linear problems and work well for many practical problems.
- The algorithm creates a line which separates the classes in case e.g., in a classification problem.
- The goal of the line is to maximizing the margin between the points on either side of the so-called decision line.
- The benefit of this process is, that after the separation, the model can easily guess the target classes (labels) for new cases.

SVM's way to find the best line

- According to the SVM algorithm we find the points closest to the line from both the classes. These points are called support vectors.
- Now, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin.
- The hyperplane for which the margin is maximum is the optimal hyperplane.
- Thus, SVM tries to make a decision boundary in such a way that the separation between the two classes (that street) is as wide as possible.

SVM using Sklearn

Model Building

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

X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

Y = data['spam'].values
```

Splitting the Dataset

```
from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split( X, Y, test_size = 0.2, random_
state = 2 )

X_train, X_test, Y_train, Y_test = train_test_split( X_train, Y_train, test_size =
0.2, random_state = 2 )
```

Standardization

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)

X_val= scaler.transform(X_val)
```

Finding the best kernel trick

```
from sklearn import svm

from sklearn.model_selection import GridSearchCV

param_grid = {

    'kernel' : ['linear', 'rbf']
}


clf = svm.SVC()

grid_search = GridSearchCV(estimator = clf, param_grid = param_grid, cv =
5)

grid_result = grid_search.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
```

```
Best: 0.974105 using {'kernel': 'linear'}
```

- Its evident that the Linear kernel gives the best accuracy.

Training the model using on the train data

```
clf = svm.SVC(kernel = 'linear')

clf.fit(X_train, Y_train)
```

Making prediction on the test data

```
Y_pred = clf.predict(X_test)
```

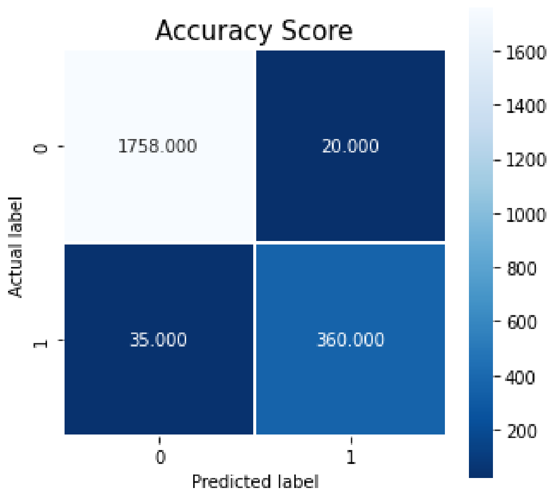
Accuracy

```
from sklearn import metrics

print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))

print("Precision:",metrics.precision_score(Y_test, Y_pred))

print("Recall:",metrics.recall_score(Y_test, Y_pred))
```

```
Accuracy: 0.9746893695352048

Precision: 0.9473684210526315

Recall: 0.9113924050632911
```

Confusion Matrix

```
plt.figure(figsize=(5, 5))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');

all_sample_title = 'Accuracy Score'

plt.title(all_sample_title, size = 15);
```



Naive Bayes

What is Naive Byes?

- Naive Bayes classifier is a classification algorithm in machine learning and is included in supervised learning.
- This algorithm is quite popular to be used in Natural Language Processing or NLP.
- This algorithm is based on the Bayes Theorem created by Thomas Bayes.
- The essence of the Bayes theorem is conditional probability where conditional probability is the probability that something will happen, given that something else has already occurred.
- By using conditional probability, we can find out the probability of an event will occur given the knowledge of the previous event.

    - P(A|B) = Posterior Probability, Probability of A given Value of B.
    - P (B|A) = Likelihood of B given A is True.
    - P (A) = Prior Probability, Probability of event A.
    - P (B) = Marginal Probability, Probability of event B.

- Some assumptions used in the Naive Bayes Classifier are that each feature in the data may not be correlated or mutually independent.

- Then the second for likelihood P (x | y) must follow one of the statistical distributions, namely Gaussian, Multinomial, or Bernoulli.

Multi-variate Bernoulli Naive Bayes

- The binomial model is useful if your feature vectors are binary (i.e., 0s and 1s).

```
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import
accuracy_score,confusion_matrix,precision_score

bnb = BernoulliNB()

bnb.fit(X_train,y_train)
y_pred3 = bnb.predict(X_test)
print(accuracy_score(y_test,y_pred3))
print(confusion_matrix(y_test,y_pred3))
print(precision_score(y_test,y_pred3))
```

```
0.9641049240681087
[[1752   26]
 [  52  343]]
0.9295392953929539
```

Multinomial Naive Bayes

- The multinomial naive Bayes model is typically used for discrete counts.

```
mnb = MultinomialNB()

mnb.fit(X_train,y_train)

y_pred2 = mnb.predict(X_test)

print(accuracy_score(y_test,y_pred2))

print(confusion_matrix(y_test,y_pred2))

print(precision_score(y_test,y_pred2))
```

```
0.970547630004602
[[1761   17]
 [  47  348]]
0.9534246575342465
```

Gaussian Naive Bayes

- In this, we assume that the features follow a normal distribution. Instead of discrete counts.

```
gnb = GaussianNB()
gnb.fit(X_train,y_train)

y_pred1 = gnb.predict(X_test)

print(accuracy_score(y_test,y_pred1))

print(confusion_matrix(y_test,y_pred1))
print(precision_score(y_test,y_pred1))
```

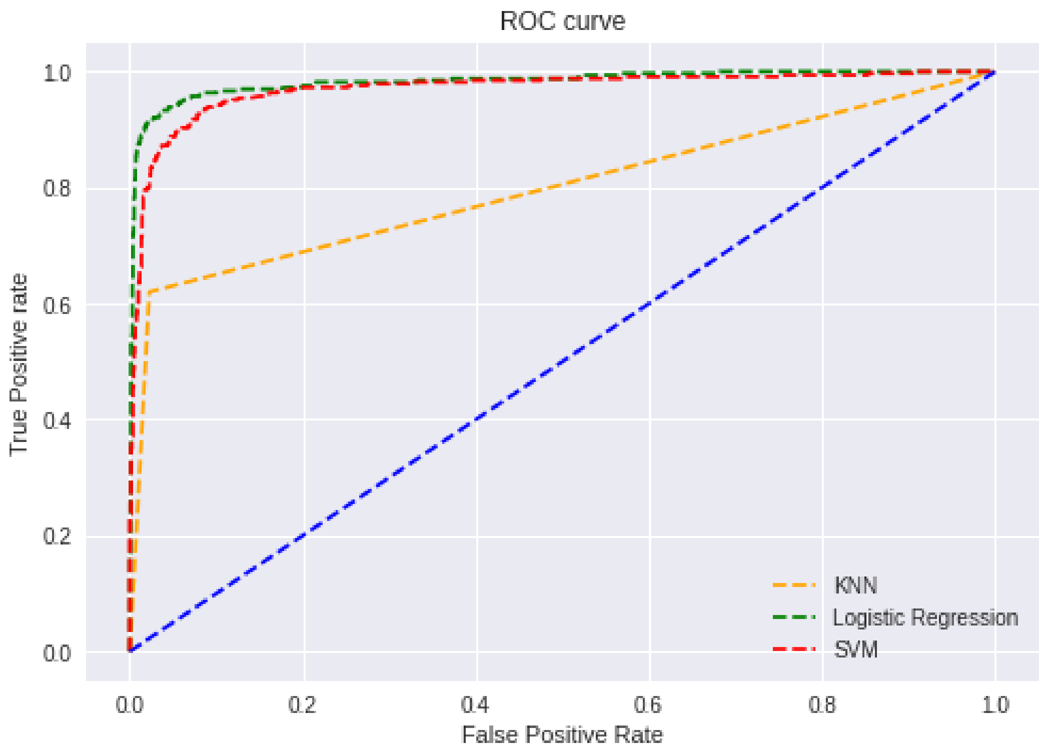
```
0.7390704095720203
[[1246  532]
 [  35  360]]
0.40358744394618834
```

AUC – ROC

```
from sklearn.metrics import roc_curve

# ROC curve for models

fpr1, tpr1, thresh1 = roc_curve(Y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(Y_test, pred_prob2[:,1], pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(Y_test, pred_prob3[:,1], pos_label=1)

# ROC curve for tpr = fpr

random_probs = [0 for i in range(len(Y_test))]
p_fpr, p_tpr, _ = roc_curve(Y_test, random_probs, pos_label=1)
from sklearn.metrics import roc_auc_score

# AUC scores

auc_score1 = roc_auc_score(Y_test, pred_prob1[:,1])
auc_score2 = roc_auc_score(Y_test, pred_prob2[:,1])
auc_score3 = roc_auc_score(Y_test, pred_prob3[:,1])

print("KNN :", auc_score1, "\nLogistic Regression :", auc_score2, "\nSVM :", au
c_score3)
```

```
KNN : 0.7986748238095958

Logistic Regression : 0.9827282709287452

SVM : 0.970177603379059
```

```
import matplotlib.pyplot as plt


plt.style.use('seaborn')
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='KNN')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='Logistic Regression')
plt.plot(fpr3, tpr3, linestyle='--',color='red', label='SVM')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show();
```



Classification Algorithms Comparison

```
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

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)
```

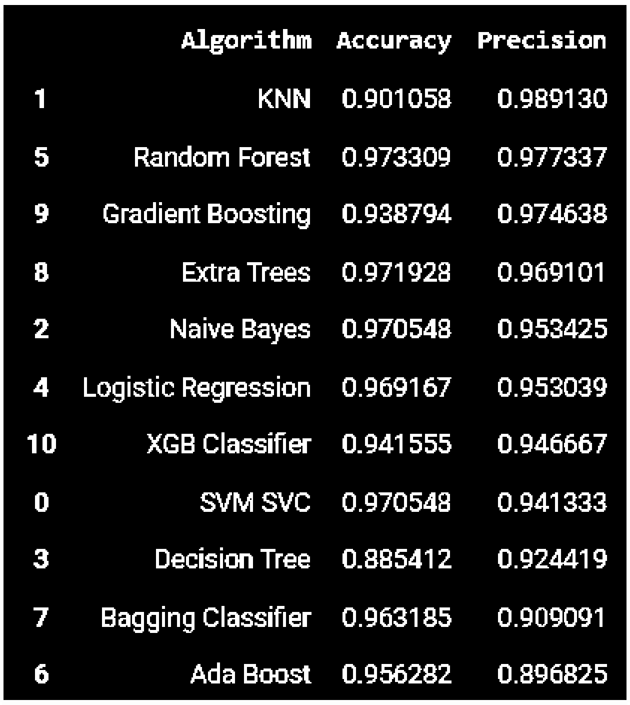
```
clfs = {
    'SVM SVC' : svc,
    'KNN' : knc,
    'Naive Bayes': mnb,
    'Decision Tree': dtc,
    'Logistic Regression': lrc,
    'Random Forest': rfc,
    'Ada Boost': abc,
    'Bagging Classifier': bc,
    'Extra Trees': etc,
    'Gradient Boosting':gbdt,
    'XGB Classifier':xgb
}
def train_classifier(clf, X_train, y_train, X_test, y_test):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    return accuracy, precision

accuracy_scores = []
precision_scores = []
for name, clf in clfs.items():
    current_accuracy, current_precision = train_classifier(clf, X_train, y_train, X_test,
y_test)
```

```
    print("For ", name)

    print("Accuracy - ", current_accuracy)

    print("Precision - ", current_precision, "\n")

    accuracy_scores.append(current_accuracy)

    precision_scores.append(current_precision)
```

```
For SVM SVC
Accuracy - 0.970547630004602
Precision - 0.9413333333333334

For KNN
Accuracy - 0.9010584445467096
Precision - 0.9891304347826086


For  Naive Bayes
Accuracy -  0.970547630004602
Precision -  0.9534246575342465

For  Decision Tree
Accuracy -  0.8854118729866544
Precision -  0.9244186046511628

For  Logistic Regression
Accuracy -  0.9691670501610676
Precision -  0.9530386740331491


For Random Forest
Accuracy -  0.9733087896916705
Precision -  0.9773371104815864
```

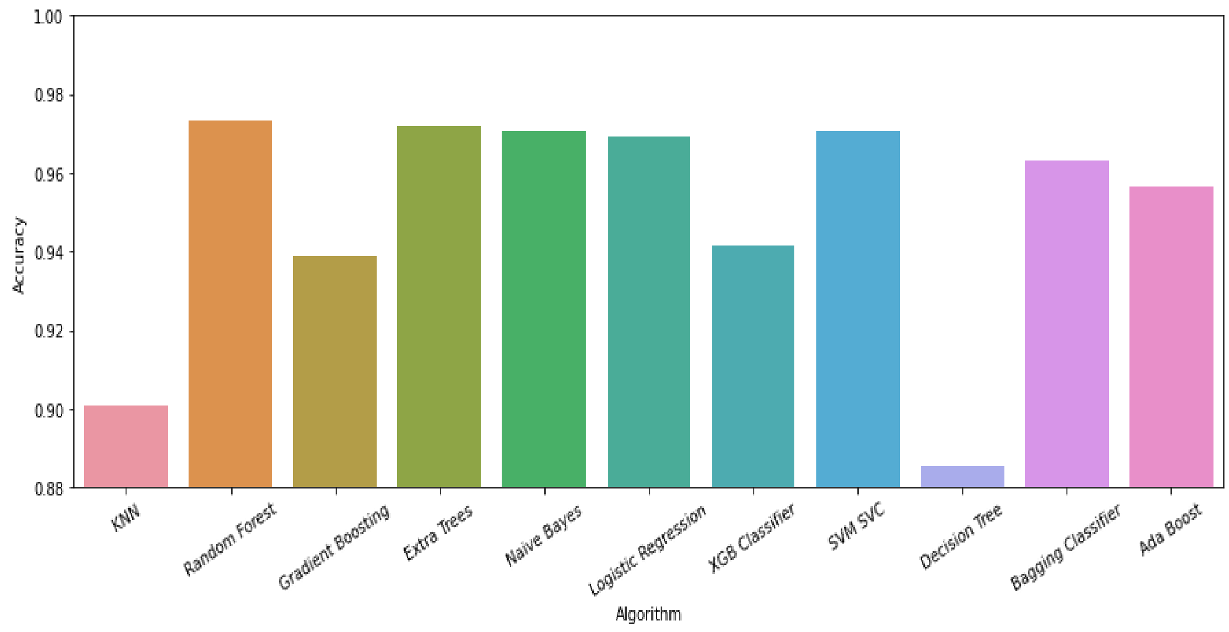
```
For Ada Boost
Accuracy -  0.956281638288081
Precision -  0.8968253968253969

For Bagging Classifier
Accuracy -  0.9631845375057524
Precision -  0.9090909090909091

For Extra Trees
Accuracy -  0.9719282098481362
Precision -  0.9691011235955056

For Gradient Boosting
Accuracy -  0.9387942936033133
Precision -  0.9746376811594203


For XGB Classifier
Accuracy - 0.9415554532903819
Precision - 0.9466666666666667
```

```
performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_
scores,'Precision':precision_scores}).sort_values('Precision',ascending=False)

performance_df
```

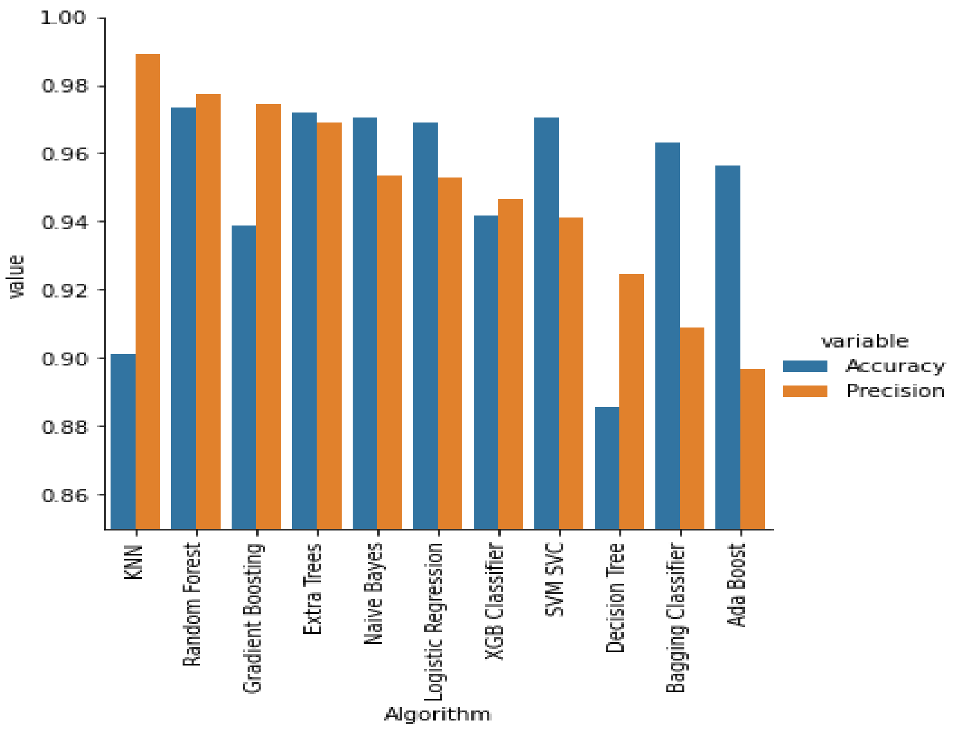


	Algorithm	Accuracy	Precision
1	KNN	0.901058	0.989130
5	Random Forest	0.973309	0.977337
9	Gradient Boosting	0.938794	0.974638
8	Extra Trees	0.971928	0.969101
2	Naive Bayes	0.970548	0.953425
4	Logistic Regression	0.969167	0.953039
10	XGB Classifier	0.941555	0.946667
0	SVM SVC	0.970548	0.941333
3	Decision Tree	0.885412	0.924419
7	Bagging Classifier	0.963185	0.909091
6	Ada Boost	0.956282	0.896825

```
plt.figure(figsize=(15, 5));

sns.barplot(x = 'Algorithm', y ='Accuracy', data = performance_df);

plt.ylim(0.88, 1.0);

plt.xticks(rotation = 30);
```

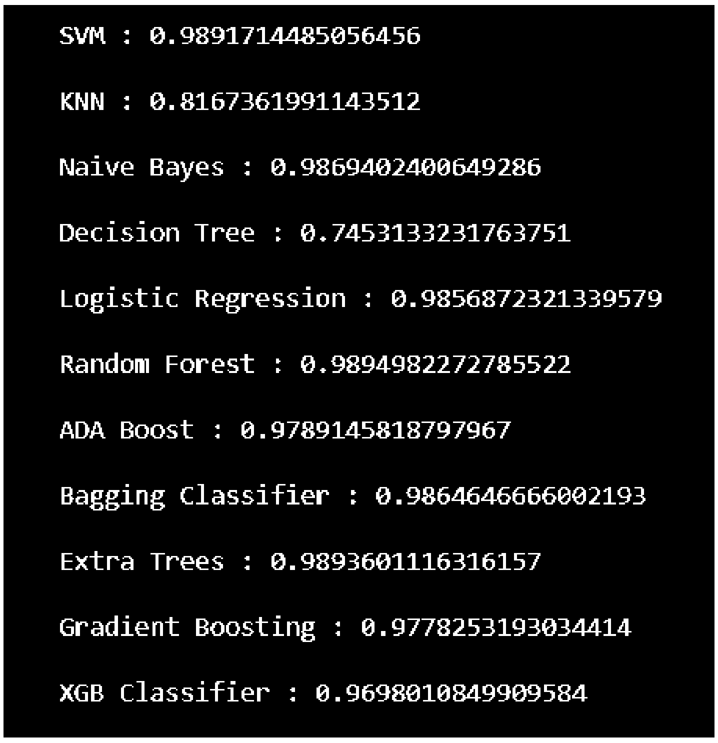


```
plt.figure(figsize=(35, 0.5));

sns.catplot(x = 'Algorithm', y='value', hue = 'variable', data=performance_df1,
kind='bar')

plt.ylim(0.85,1.0);

plt.xticks(rotation = 'vertical');
```



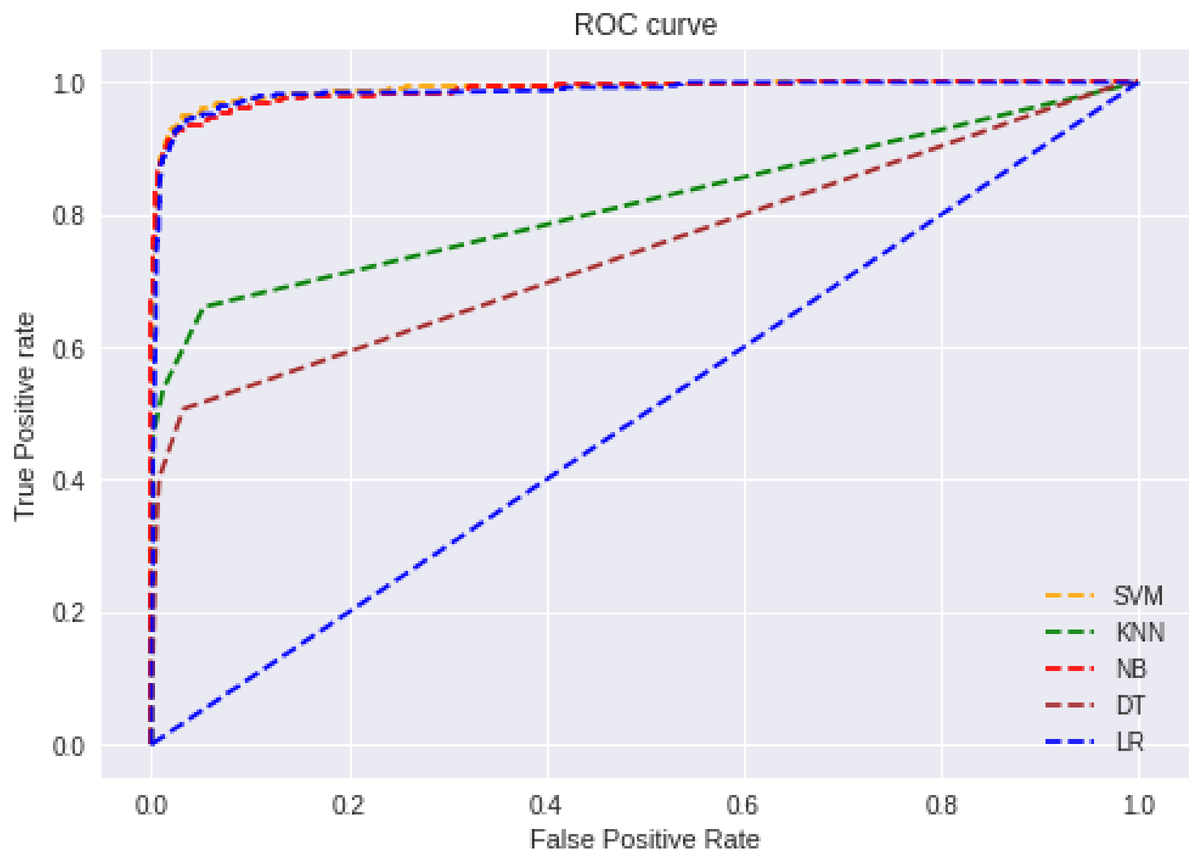
```
from sklearn.metrics import roc_curve

# ROC curve for models

fpr1, tpr1, thresh1 = roc_curve(y_test, pred_proba1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_proba2[:,1], pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(y_test, pred_proba3[:,1], pos_label=1)
fpr4, tpr4, thresh4 = roc_curve(y_test, pred_proba4[:,1], pos_label=1)
fpr5, tpr5, thresh5 = roc_curve(y_test, pred_proba5[:,1], pos_label=1)
fpr6, tpr6, thresh6 = roc_curve(y_test, pred_proba6[:,1], pos_label=1)
fpr7, tpr7, thresh7 = roc_curve(y_test, pred_proba7[:,1], pos_label=1)
```

```
fpr8, tpr8, thresh8 = roc_curve(y_test, pred_proba8[:,1], pos_label=1)
fpr9, tpr9, thresh9 = roc_curve(y_test, pred_proba9[:,1], pos_label=1)
fpr10, tpr10, thresh10 = roc_curve(y_test, pred_proba10[:,1], pos_label=1)
fpr11, tpr11, thresh11 = roc_curve(y_test, pred_proba11[:,1], pos_label=1)

# ROC curve for tpr = fpr

random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
from sklearn.metrics import roc_auc_score

# AUC scores

auc_score1 = roc_auc_score(y_test, pred_proba1[:,1])
auc_score2 = roc_auc_score(y_test, pred_proba2[:,1])
auc_score3 = roc_auc_score(y_test, pred_proba3[:,1])
auc_score4 = roc_auc_score(y_test, pred_proba4[:,1])
auc_score5 = roc_auc_score(y_test, pred_proba5[:,1])
auc_score6 = roc_auc_score(y_test, pred_proba6[:,1])
auc_score7 = roc_auc_score(y_test, pred_proba7[:,1])
auc_score8 = roc_auc_score(y_test, pred_proba8[:,1])
auc_score9 = roc_auc_score(y_test, pred_proba9[:,1])
auc_score10 = roc_auc_score(y_test, pred_proba10[:,1])
auc_score11 = roc_auc_score(y_test, pred_proba11[:,1])

print("SVM :", auc_score1)
print("\nKNN :", auc_score2)
print("\nNaive Bayes :", auc_score3)
print("\nDecision Tree :", auc_score4)
print("\nLogistic Regression :", auc_score5)
print("\nRandom Forest :", auc_score6)
print("\nADA Boost :", auc_score7)
print("\nBagging Classifier :", auc_score8)
print("\nExtra Trees :", auc_score9)
print("\nGradient Boosting :", auc_score10)
print("\nXGB Classifier :", auc_score11)
```

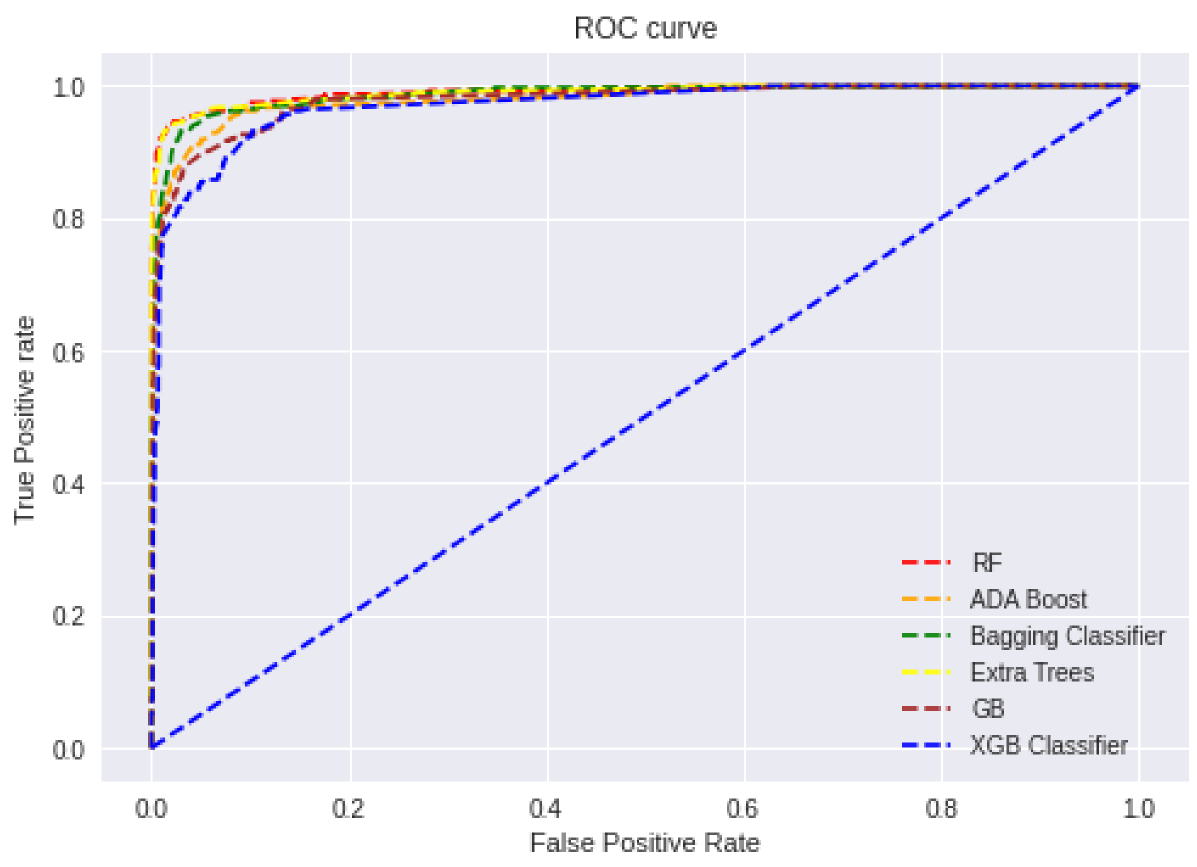


```
SVM : 0.9891714485056456

KNN : 0.8167361991143512

Naive Bayes : 0.9869402400649286

Decision Tree : 0.7453133231763751

Logistic Regression : 0.9856872321339579

Random Forest : 0.9894982272785522

ADA Boost : 0.9789145818797967

Bagging Classifier : 0.9864646666002193

Extra Trees : 0.9893601116316157

Gradient Boosting : 0.9778253193034414

XGB Classifier : 0.9698010849909584
```

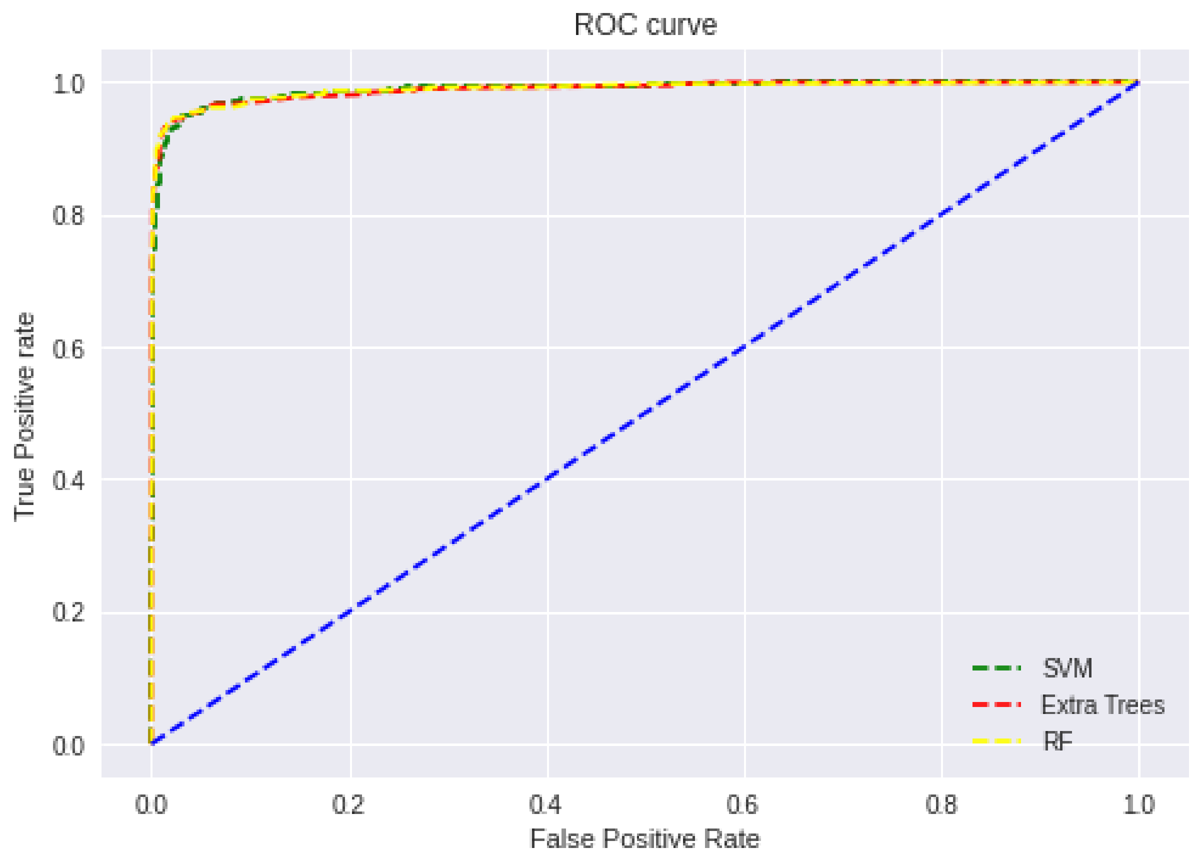
```
import matplotlib.pyplot as plt

plt.style.use('seaborn')

plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='SVM')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(fpr3, tpr3, linestyle='--',color='red', label='NB')
plt.plot(fpr4, tpr4, linestyle='--',color='brown', label='DT')
plt.plot(fpr5, tpr5, linestyle='--',color='blue', label='LR')

plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show();
```



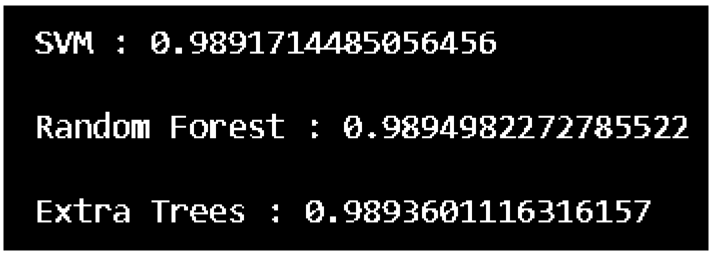
```
plt.plot(fpr6, tpr6, linestyle='--',color='red', label='RF')
plt.plot(fpr7, tpr7, linestyle='--',color='orange', label='ADA Boost')
plt.plot(fpr8, tpr8, linestyle='--',color='green', label='Bagging Classifier')
plt.plot(fpr9, tpr9, linestyle='--',color='yellow', label='Extra Trees')
plt.plot(fpr10, tpr10, linestyle='--',color='brown', label='GB')
plt.plot(fpr11, tpr11, linestyle='--',color='blue', label='XGB Classifier')

plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')


plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show();
```



```
plt.plot(fpr1, tpr1, linestyle='--',color='green', label='SVM')
plt.plot(fpr9, tpr9, linestyle='--',color='red', label='Extra Trees')
plt.plot(fpr6, tpr6, linestyle='--',color='yellow', label='RF')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show();
```



```
print("SVM :", auc_score1)

print("\nRandom Forest :", auc_score6)

print("\nExtra Trees :", auc_score9)
```



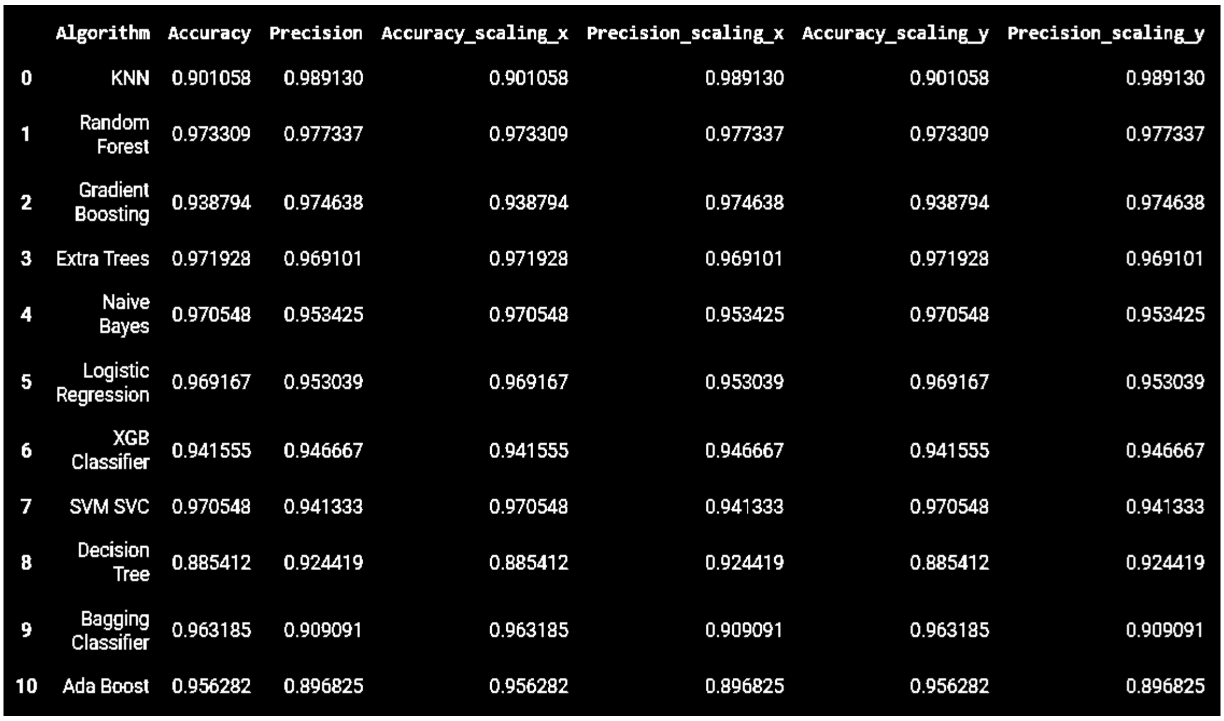
```
SVM : 0.9891714485056456

Random Forest : 0.9894982272785522

Extra Trees : 0.9893601116316157
```

- We can see from the above graph that the Random Forest model captures the highest AUC and can be considered as the best performing model among all models.

- We can see a healthy ROC curve, pushed towards the top-left side both for positive and negative classes.

Improving the Model

```
temp_df = pd.DataFrame ({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000' :
accuracy_scores,'Precision_max_ft_3000' : precision_scores}).sort_values
('Precision_max_ft_3000',ascending=False)


temp_df = pd.DataFrame ({'Algorithm':clfs.keys(),'Accuracy_scaling':
accuracy_scores, 'Precision_scaling':precision_scores}).sort_values
('Precision_scaling',ascending=False)

new_df = performance_df.merge(temp_df,on='Algorithm')
new_df_scaled = new_df.merge(temp_df,on='Algorithm')

temp_df = pd.DataFrame ({'Algorithm':clfs.keys(),'Accuracy_num_chars':
accuracy_scores,'Precision_num_chars': precision_scores}).sort_values
('Precision_num_chars', ascending=False)

new_df_scaled.merge(temp_df,on='Algorithm')
```



	Algorithm	Accuracy	Precision	Accuracy_scaling_x	Precision_scaling_x	Accuracy_scaling_y	Precision_scaling_y
0	KNN	0.901058	0.989130	0.901058	0.989130	0.901058	0.989130
1	Random Forest	0.973309	0.977337	0.973309	0.977337	0.973309	0.977337
2	Gradient Boosting	0.938794	0.974638	0.938794	0.974638	0.938794	0.974638
3	Extra Trees	0.971928	0.969101	0.971928	0.969101	0.971928	0.969101
4	Naive Bayes	0.970548	0.953425	0.970548	0.953425	0.970548	0.953425
5	Logistic Regression	0.969167	0.953039	0.969167	0.953039	0.969167	0.953039
6	XGB Classifier	0.941555	0.946667	0.941555	0.946667	0.941555	0.946667
7	SVM SVC	0.970548	0.941333	0.970548	0.941333	0.970548	0.941333
8	Decision Tree	0.885412	0.924419	0.885412	0.924419	0.885412	0.924419
9	Bagging Classifier	0.963185	0.909091	0.963185	0.909091	0.963185	0.909091
10	Ada Boost	0.956282	0.896825	0.956282	0.896825	0.956282	0.896825

Python Packages Used

Core Data Handling Packages

NumPy

- Python has a strong set of data types and data structures. Yet it wasn’t designed for Machine Learning per say.
- Numpy is a data handling library, particularly one which allows us to handle large multi-dimensional arrays along with a huge collection of mathematical operations.
- Numpy isn’t just a data handling library known for its capability to handle multidimensional data.

- It is also known for its speed of execution and vectorization capabilities.
- It provides MATLAB style functionality and it is also a core dependency for other majorly used libraries like pandas, matplotlib and so on.

Advantages

- Matrix (and multi-dimensional array) manipulation capabilities like transpose, reshape, etc.
- Highly efficient data-structures which boost performance and handle garbage collection with a breeze.
- Capability to vectorize operation, again improves performance and parallelization capabilities.

Disadvantages

- Its high performance comes at a cost. The data types are native to hardware and not python, thus incurring an overhead when numpy objects have to be transformed back to python equivalent ones and vice-versa.

Pandas

- Pandas is a python library that provides flexible and expressive data structures (like dataframes and series) for data manipulation.
- It is built on top of numpy, pandas is as fast and yet easier to use. Pandas provides capabilities to read and write data from different sources like CSVs, Excel, SQL Databases, HDFS and many more.
- It provides functionality to add, update and delete columns, combine or split dataframes/series, handle datetime objects, impute null/missing values, handle time series data, conversion to and from numpy objects and so on.

Advantages

- Extremely easy to use and with a small learning curve to handle tabular data.
- Amazing set of utilities to load, transform and write data to multiple formats.
- Compatible with underlying NumPy objects and go to choice for most Machine Learning libraries like scikit-learn, etc.
- Capability to prepare plots/visualizations out of the box (utilizes matplotlib to prepare different visualization under the hood).

Disadvantages

- The ease of use comes at the cost of higher memory utilization.
- Pandas creates far too many additional objects to provide quick access and ease of manipulation.
- Inability to utilize distributed infrastructure.
- Though pandas can work with formats like HDFS files, it cannot utilize distributed system architecture to improve performance.

Scipy

- Scipy is one of the most important python libraries of all time.
- Scipy is a scientific computing library for python.
- It is also built on top of numpy and is a part of the Scipy Stack.
- This is yet another behind the scenes library which does a whole lot of heavy lifting.
- It provides modules/algorithms for linear algebra, integration, image processing, optimizations, clustering, sparse matrix manipulation and many more.

Collections

- The Python collection module is defined as a container that is used to store collections of data such as list, dictionary, set, and tuple, etc.

- It was introduced to improve the functionalities of the built-in collection.

PIL

- The Python Imaging Library adds image processing capabilities to your Python interpreter.
- This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.
- The core image library is designed for fast access to data stored in a few basic pixel formats. It provides a solid foundation for a general image processing tool.
- It is the essential modules for image processing in Python.
- It supports the variability of images such as jpeg, png, bmp, gif, ppm, and tiff.

Disadvantages

- It is not supported by Python 3.

Matplotlib

- Another component of the SciPy stack, matplotlib is essentially a visualization library.
- It works seamlessly with NumPy objects (and its high-level derivatives like pandas). Matplotlib provides a MATLAB like plotting environment to prepare high-quality figures/charts for publications, notebooks, web applications and so on.
- Matplotlib is a high customizable low-level library that provides a whole lot of controls and knobs to prepare any type of visualization/figure.
- Given its low-level nature, it requires a bit of getting used to along with plenty of code to get stuff done.
- It’s well documented and extensible design has allowed a whole list of high-level visualization libraries to be built on top.

Advantages

- Extremely expressive and precise syntax to generate highly customizable plots
- Can be easily used in line with Jupyter notebooks.

Disadvantages

- Heavy reliance on numpy and other Scipy stack libraries
- Huge learning curve, it requires quite a bit of understanding and practice to use matplotlib.

Scikit-learn

- Scikit-learn provides a simple yet powerful fit-transform and predict paradigm to learn from data, transform the data and finally predict.
- Using this interface, it provides capabilities to prepare classification, regression, clustering and ensemble models.
- It also provides a multitude of utilities for pre-processing, metrics, model evaluation techniques, etc.

Advantages

- The go-to package that has it all for classical Machine Learning algorithms.
- Consistent and easy to understand interface of fit and transform.
- Capability to prepare pipelines not only helps with rapid prototyping but also quick and reliable deployments.

Disadvantages

- Inability to utilize categorical data for algorithms out of the box that support such data types (packages in R have such capabilities)
- Heavy reliance on the Scipy stack.

Math

- This is the most basic math module that is available in Python. It covers basic mathematical operations like sum, exponential, modulus, etc.
- This library is not useful when dealing with complex mathematical operations like multiplication of matrices.
- The calculations performed with the functions of the python math library are also much slower.
- However, this library is adequate when you have to carry out basic mathematical operations.

NLP Packages

Natural Language Toolkit (NLTK)

- The list includes low-level tasks such as tokenization (it provides different tokenizers), n-gram analysers, collocation parsers, POS taggers, NER and many more.
- NLTK is primarily for English based NLP tasks.
- It is widely used in academic and industrial institutions across the world.

Advantages

- The goto library for most NLP related tasks.
- Provides a huge array of algorithms and utilities to handle NLP tasks, right from low-level parsing utilities to high-level algorithms like CRFs.
- Extensible interface which allows us to train and even extend existing functions and algorithms.

Disadvantages

- Mostly written in java, it has overheads and limitations in terms of the amount of memory required to handle huge datasets

- Inability to interface with the latest advancements in NLP using deep learning models

Visualization Packages

Seaborn

- Built on top of matplotlib, seaborn is a high-level visualization library.
- It provides sophisticated styles straight out of the box (which would take some good amount of effort if done using matplotlib).
- Apart from styling prowess and sophisticated color pallets, seaborn provides a range of visualizations and capabilities to work with multivariate analysis.
- It provides capabilities to perform regression analysis, handling of categorical variables and aggregate statistics.

Word Cloud

- A tag cloud (word cloud or wordle or weighted list in visual design) is a novelty visual representation of text data, typically used to depict keyword metadata (tags) on websites, or to visualize free form text.
- Tags are usually single words, and the importance of each tag is shown with font size or color.

- A word cloud lets us easily identify the keywords in a text where the size of the words represents their frequency.
- With this, we’ll get a good idea of what a text is about before even reading it.

Boosting Packages

XGBoost

- One of the most widely used libraries/algorithms used in various data science competitions and real-world use cases, XGBoost is probably one of the best-known variants.
- A highly optimized and distributed implementation, XGBoost enables parallel execution and thus provides immense performance improvement over gradient boosted trees.
- It provides capabilities to execute over distributed frameworks like Hadoop easily.

Miscellaneous Packages

IPython and Jupyter

- IPython or Interactive Python is a command line interface which supports parallel computing and a host of GUI toolkits and also forms the core of web application-based notebook server called Jupyter.
- It allows us to prepare and share documents which contain live code, interactive visualizations, markdown and slideshow capabilities.

Data Interpretation

- ❖ Original Data Set : Drive
- ❖ Processed Data Set : Drive

Source Code : Colab

GitHub Gist : Gist

Thankyou!!