

SMS Spam Detection

IDS Project Report submitted by

Team: 8

21ucc044- Harsh Bansal

21ucs056- Dev Mittal

21ucs054- Daksh Bansal

21ucs127- Mehul Khera

[Github Link](#)

Course Instructors:

Dr. Lal Upendra Pratap Singh

Dr. Subrat K. Dash

Dr. Alope Datta

Department of Computer Science & Engineering
The LNM Institute of Information Technology, Jaipur

OBJECTIVE

To collect the dataset from the website and perform the following operations:

- Preprocessing of the data and visualizing it
- Getting inferences out of it
- Explanation and Implementation of ML Classification Algorithms
- Getting results from the test data and visualization from various means

SOURCE OF DATASET

The dataset is sourced from the **UCI ML repository** and can be found [here](#).

INTRODUCTION TO DATASET

A collection of 425 SMS spam messages was manually extracted from the [Grumbletext Web site](#). This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involves carefully scanning hundreds of web pages.

A subset of 3,375 SMS randomly chosen ham messages from the **NUS SMS Corpus (NSC)**, which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore, The messages largely originate from Singaporeans and mostly from students attending the university. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

DATASET DESCRIPTION AND FORMAT:

The dataset contains one message per line. Each line is composed of two columns: one with a label (ham or spam) and the second column with the raw text. Here are some examples:

ham What you doing?how are you?

ham Ok lar... Joking wif u oni...

ham dun say so early hor... U c already then say...

ham MY NO. IN LUTON 0125698789 RING ME IF UR AROUND! H*

ham Siva is in hostel aha:-.

ham Cos i was out shopping wif darren jus now n i called him 2 ask wat present he wan lor. Then he started guessing who i was wif n he finally guessed darren lor.

spam FreeMsg: Txt: CALL to No: 86888 & claim your reward of 3 hours talk time to use from your phone now! ubscribe6GBP/ mnth inc 3hrs 16 stop?txtStop

spam Sunshine Quiz! Win a super Sony DVD recorder if you canname the capital of Australia? Text MQUIZ to 82277. B

spam URGENT! Your Mobile No 07808726822 was awarded a L2,000 Bonus Caller Prize on 02/09/03! This is our 2nd attempt to contact YOU! Call 0871-872-9758 BOX95QU

Note: These messages are not chronologically sorted.

Due to the lack of organization of the features, we needed to preprocess the data and extract the required information. Further information is discussed later.

IMPLEMENTATION DETAILS

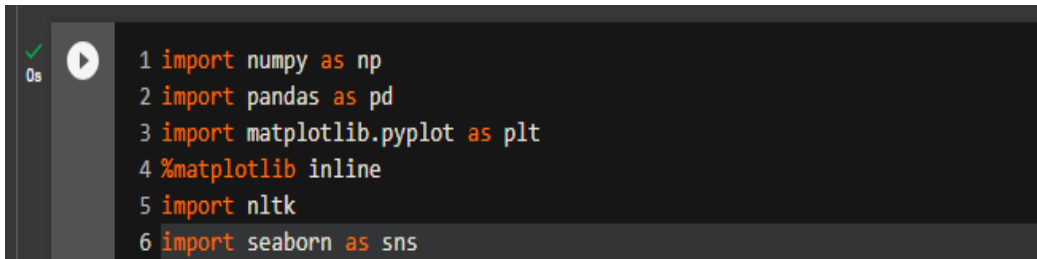
A. Connecting to Google Colab

This part is specific to the platform being used.

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

We used Google Colab to facilitate faster and more efficient running of the code. **We first uploaded the data to Gdrive, and from there, we mounted our drive onto this platform by giving the appropriate address.**

B. Importing the Libraries



```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5 import nltk
6 import seaborn as sns
```

- a. **Numpy:** NumPy is the fundamental package for scientific computing in Python. NumPy aims to provide an array object that is up to 50 times faster than traditional Python lists.
- b. **Pandas:** The Pandas library provides high-performance, easy-to-use data structures. We used the pandas' library to import the dataset in our data frame, which was used in all the further steps.
- c. **Matplotlib:** Matplotlib is an amazing visualization library in Python for 2D plots of arrays. One of the greatest benefits of visualization is that it allows us to visually access huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like lines, bars, scatters, histograms, etc.
- d. **Seaborn:** Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- e. **nltk:** the Python Natural Language Toolkit is a robust library designed for natural language processing (NLP) tasks. It offers a comprehensive set of tools, from basic tokenization to advanced tasks like part-of-speech tagging and parsing. NLTK stands out for its extensive collection of corpora and lexical resources, making it a preferred choice for language analysis. With a user-friendly interface, it supports text classification, sentiment analysis, and machine learning with textual data. Widely used by researchers, educators, and developers, NLTK's open-source nature and seamless integration with other Python libraries make it a go-to solution for exploring the intricacies of linguistic data.

C. Importing the Dataset

We have imported the dataset as a data frame of the pandas' library in Python to make further operations fast and easy.

```
1 column_names=['A','B']
2 df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/SMS SpamCollection.csv",
3               on_bad_lines='skip',names=column_names)
4 df.head()
5
```

	A	B
0	ham\tGo until jurong point	crazy.. Available only in bugis n great world...
1	ham\tOK lar... Joking wif u oni...	NaN
2	spam\tFree entry in 2 a wkly comp to win FA Cu...	NaN
3	ham\tU dun say so early hor... U c already the...	NaN
4	ham\tNah I don't think he goes to usf	he lives around here though

```
[64] 1 df=df.drop('B',axis=1) # to remove the unwanted series no column
      2 df.head()
```

	A
0	ham\tGo until jurong point
1	ham\tOK lar... Joking wif u oni...
2	spam\tFree entry in 2 a wkly comp to win FA Cu...
3	ham\tU dun say so early hor... U c already the...
4	ham\tNah I don't think he goes to usf

D. Data Cleaning

a. Remove unnecessary features or columns for the dataset:

As given in the above picture, on reading the CSV file directly from the dataset, we were getting an unnecessary column of B due to the long text. Thus, we have dropped the B column from the 1st axis.

b. Data Cleaning and Extraction:

We can observe that the text message and text type (whether ham or spam) are found as a single string for each row; we need to separate them as two separate columns: 'Target' for message type and 'Text' for message text. We have used '\t' as a delimiter to separate the data frames. With that, we have removed the previous field of 'A' of raw data and will work on the two columns produced here from now on.

DATA CLEANING

```
[81] 1 df[['Target', 'Text']] = df['A'].str.split('\t', expand=True)
      2 #Creating two seprate
      3 df.head()
      4
```

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

```
[82] 1 df.drop('A', axis=1, inplace=True)
      2 df.head()
```

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

```
[83] 1 df.shape
```

(5205, 2)

c. **Using Label Encoder to convert 'HAM' and 'SPAM' into binary encoding:**

It is generally better to process the data as numerically as possible for better processing as compared to categorical values. Thus, we will encode the HAM -> 0
SPAM ->1

Using Label Encoder to convert ham and spam into binary encoding

```
[50] 1 from sklearn.preprocessing import LabelEncoder
      2 lab=LabelEncoder()
      3 df['Target']=lab.fit_transform(df['Target'])
      4 df.head()
```

	Target	Text
0	0	Go until jurong point
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf

We are using the preprocessing library of the sci-kit-learn library, a widely used machine learning toolkit in Python, to encode our labels. The resulting encoded labels replace the original categorical values in the 'Target' column.

d. Checking for null values:

While making a data frame from a.csv file, many blank columns are imported as null values into the data frame, which later creates problems while operating that data frame. Hence, the `isnull()` method is used to check for NULL values in the data frame.

Below is the code snippet for the same:

```
[51] 1 df.isnull().sum()#checking for missing values

      Target    0
      Text     0
      dtype: int64
```

As we can see from the above output, there are no NULL values present in our dataset.

e. Dealing with the duplicates:

Often, we can have redundant data entries, which can decrease the quality of the data and increase the overhead for decreased computation speed. Thus, we will first count the duplicates and drop the duplicates while keeping only their first choice.

```
[52] 1 #count number of duplicate values
      2 df.duplicated().sum()

      483

[53] 1 #drop duplicates but keep only their first copies
      2 df=df.drop_duplicates(keep='first')

[54] 1 df.shape

      (4722, 2)
```

EXPLORING DATA ANALYSIS AND VISUALIZATION

a. Basic information about our preprocessed data:

```
[ ] 1 df.info()

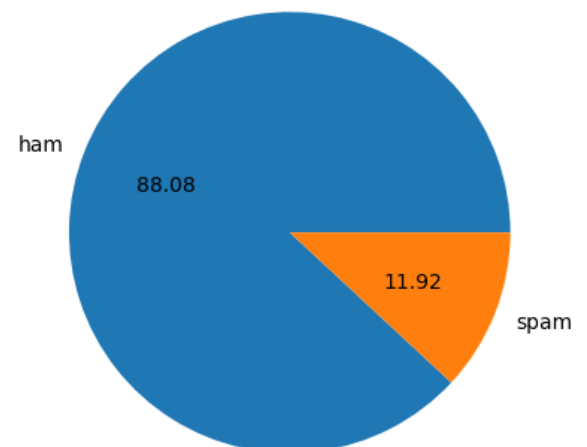
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4722 entries, 0 to 5204
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0    Target    4722 non-null    int64
1    Text      4722 non-null    object
dtypes: int64(1), object(1)
memory usage: 110.7+ KB
```

Basic information about the data frames in our set. The `df.info()` method provides a holistic view of the data frame, encompassing its structure, data types, memory usage, and potential data quality issues. It's a valuable tool for initial data exploration and understanding the characteristics of the data you're working with.

b. Piechart of the message type

Pie charts play a crucial role in data analysis by visually presenting proportions within a dataset in a simple, accessible manner. Their circular format effectively communicates the distribution of a whole into its parts, making it easy to identify major contributors or trends. Here, we can analyze the percentage of spam messages in our dataset. As we can observe, there is almost 12% of spam messages in our dataset.

Percentage of SPAM messages



```
1 grouping=df.groupby('Target')
2 df['Target'].value_counts()
3 plt.pie(df['Target'].value_counts(),labels=['ham','spam'],autopct='%0.2f')
4 plt.title('Percentage of SPAM messages')
5 plt.show()
```


c. Adding extra features after analysis: Feature Engineering

It is part of preprocessing; it would be figured when analyzing the data and adding some columns into the dataset, which includes

1. num_words, i.e. the total number of words
2. num_char, i.e. the total number of characters in a text message
3. num_sent, i.e., the total number of sentences, we will have the full stop as a delimiter.

We will use the functions of nltk library for the natural language processing library of text data.

```
1 #nltk.download('punkt')
2 df['num_char']=df['Text'].apply(len)
3 df['num_words']=df['Text'].apply(lambda x: len(nltk.word_tokenize(x)))
4 df['num_sent']=df['Text'].apply(lambda x: len(nltk.sent_tokenize(x)))
5 df
```

1 to 10 of 4722 entries

index	Target	Text	num_char	num_words	num_sent
0	0	Go until jurong point	21	4	1
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf	32	9	1
5	1	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send	133	35	4
6	0	Even my brother is not like to speak with me. They treat me like aids patent.	77	18	2
7	0	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	160	31	2
8	1	WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.	157	32	5
9	1	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	154	31	3

Show 10 per page 1 2 10 100 400 470 473

Statistical Analysis of the Data

If we look at the overall statistical analysis of the messages, we can find that the average word count (mean) in a text message is 15 and the number of lines is 1.8, which can be approximated as 2 lines per message.

```
1 df[['num_char', 'num_words', 'num_sent']].describe()
```

	num_char	num_words	num_sent
count	4722.0	4722.0	4722.0
mean	65.98390512494706	15.365311308767472	1.8242270224481152
std	50.26253628562214	11.310214719403897	1.3054971864220812
min	0.0	0.0	0.0
25%	30.0	8.0	1.0
50%	49.0	12.0	1.0
75%	89.0	21.0	2.0
max	910.0	220.0	38.0

Show 25 per page

Now, only looking at the data from spam messages, we can find an interesting comparison between ham and spam messages.

```
1 df[df['Target']==1][['num_char', 'num_words', 'num_sent']].describe()
```

	num_char	num_words	num_sent
count	563.0	563.0	563.0
mean	120.26820603907638	23.94316163410302	2.7015985790408528
std	42.58858260294995	8.988543634909657	1.4328806322196375
min	2.0	1.0	1.0
25%	91.0	18.0	2.0
50%	137.0	27.0	3.0
75%	154.0	31.0	4.0
max	183.0	45.0	9.0

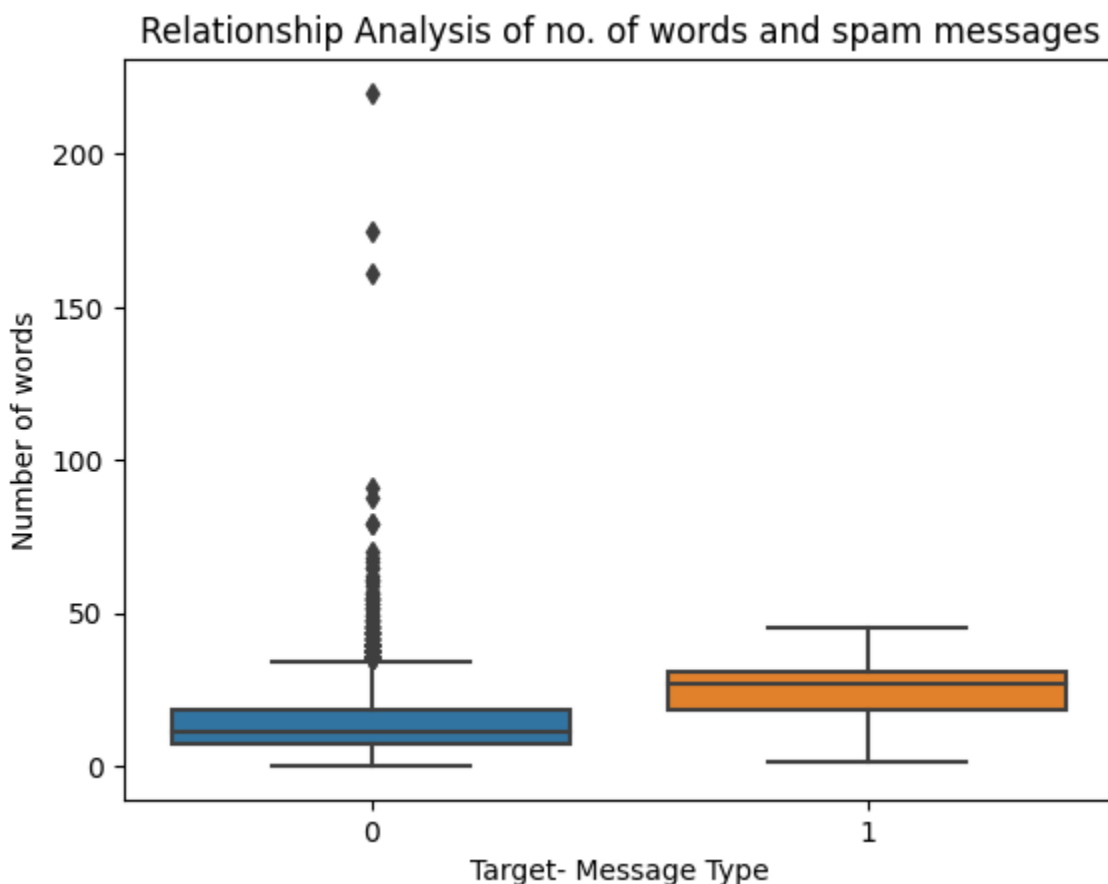
Show 25 per page

e. Relationship between the number of words and target message type:

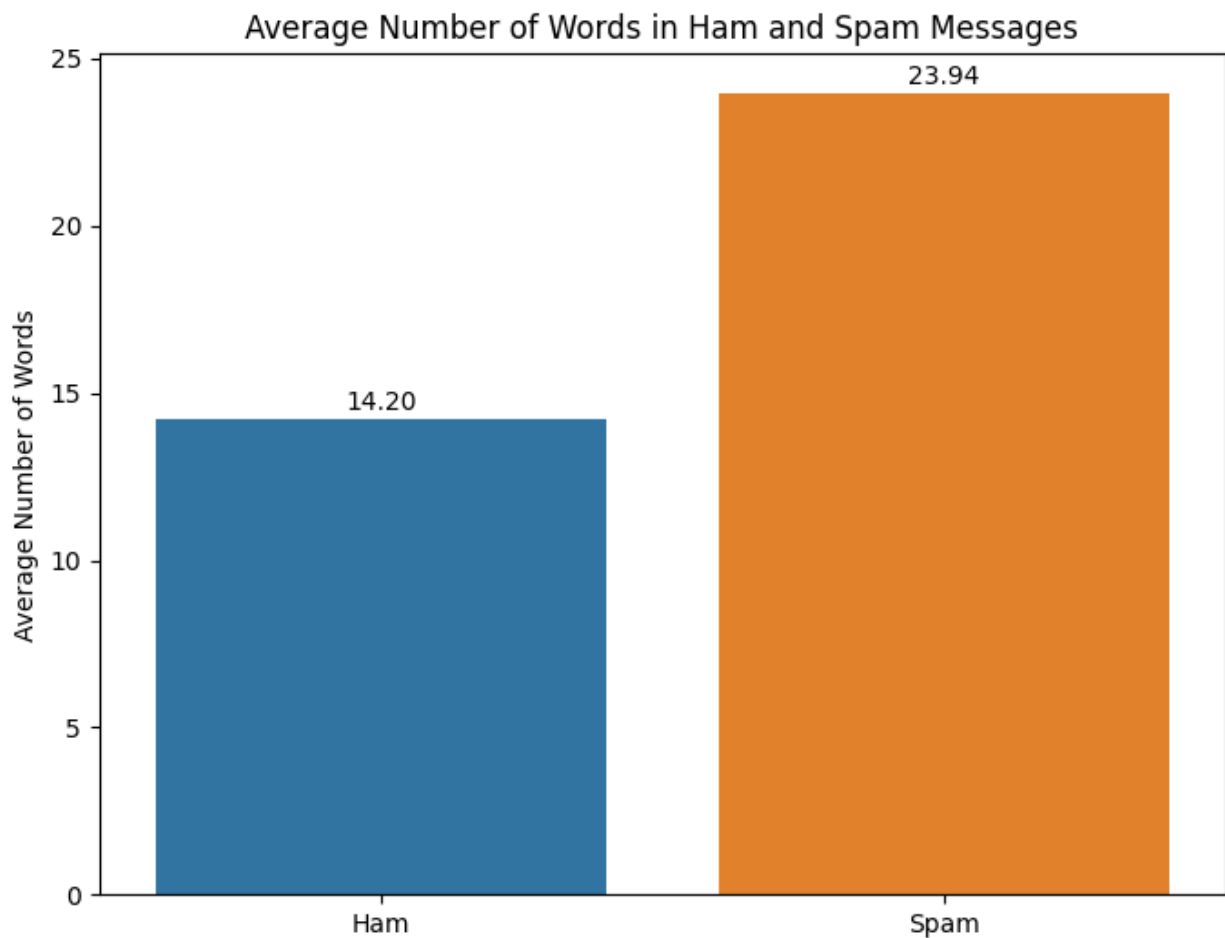
We are interested in exploring the possibility of the relationship between the number of words in a text message and the possibility of it being a spam message.

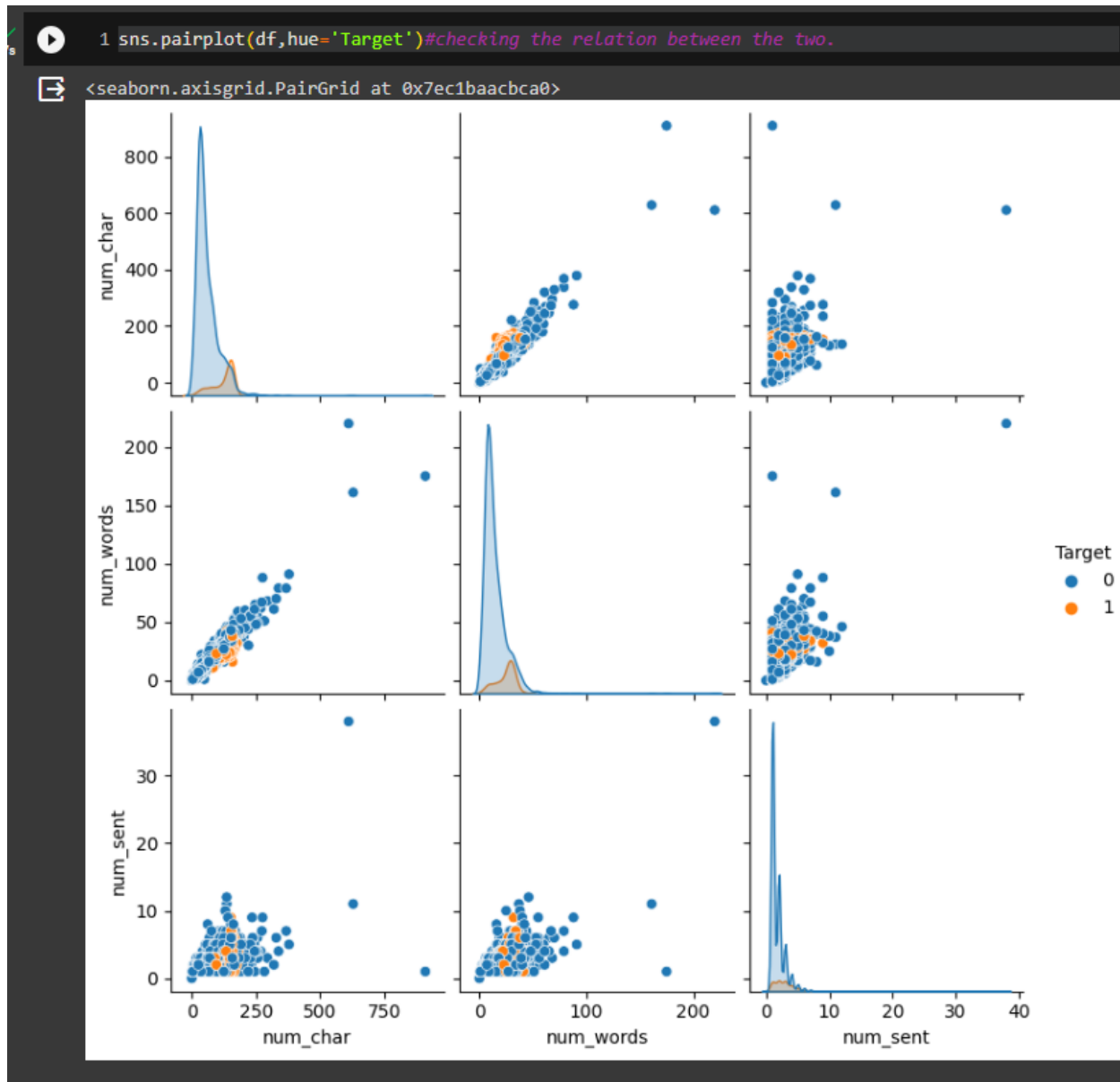
We have explored it through a box plot, A box plot is a visual representation of data distribution. It summarizes five important aspects of the data: minimum, first quartile, median, third quartile, and maximum. Box plots are useful for comparing data sets and identifying outliers.

```
1 #Box Plot of num_words and Target
2 sns.boxplot(x = 'Target', y = 'num_words', data = df)
3 plt.xlabel('Target- Message Type')
4 plt.ylabel('Number of words')
5 plt.title('Relationship Analysis of no. of words and spam messages')
6 plt.show()
```



An analysis of the average number of words transmitted and quartiles reveals that spam messages generally contain a greater quantity of words. Additionally, we should recognize that outliers are exclusive to non-spam (ham) messages and that extremely lengthy text messages can be identified in ham messages but not in spam messages.





For comprehensive data exploration and distribution, we have used the pairplot of the Seaborn library. Pairplot is a powerful data visualization tool in the Seaborn library for exploring relationships between multiple variables. It generates a matrix of plots, where each plot displays the pairwise relationship between two variables. This allows us to quickly identify patterns, trends, and outliers in our data.

DATA PREPROCESSING TO FEED INTO MODELS

Data preprocessing is essential in data science for ensuring data quality, handling missing values, standardizing numerical features, managing outliers, and encoding categorical variables. It improves the accuracy and reliability of analyses and machine learning models by addressing data issues and optimizing input for algorithms.

- a. **Tokenization:** We have used tokenization for mainly three purposes:
 1. Vocabulary Creation: To create a vocabulary, which is a unique set of all tokens present in the text. This is important for text classification, where the understanding of specific words is essential.
 2. Text Normalization: Tokenization is often combined with text normalization techniques, such as converting all text to lowercase, removing punctuation, or handling contractions. This ensures consistent and standardized representations of words.
 3. Stopword Removal: Tokenization enables the identification and removal of stopwords (common words like "the," "and", "is" etc.) during preprocessing. Stopword removal is essential for reducing noise in text data and focusing on more meaningful words.

We have utilized the Natural Language Toolkit (NLTK) library in Python for text preprocessing. It begins by importing NLTK and downloading the stopwords dataset, which consists of common words often excluded in text analysis. Subsequently, the script imports the stopwords module and the PorterStemmer class from NLTK, the latter being a tool for word stemming, reducing words to their root form. The code then creates an instance of **PorterStemmer**.

```
1 import nltk
2 nltk.download('stopwords')
3 from nltk.corpus import stopwords
4 from nltk.stem.porter import PorterStemmer
5 ps=PorterStemmer()
6 #first import porter stemmer---present few cells below before running this cell
```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

```
[306] 1 import string
2 punc=string.punctuation
3 def preprocess(text):#function to lowercase,tokenize,retain only alphanumeric values, stemming
4     text=text.lower()
5     text=nlTK.word_tokenize(text)
6     [word for word in text if word not in punc]
7     arr=[]
8     for i in text:
9         if i.isalnum():
10             arr.append(i)
11     text=arr[:]
12     arr.clear()
13     '''need to clear the arr everytime before applying another function
14     so that the processed text doesnt get appended multiple times'''
15     for i in text:
16         arr.append(ps.stem(i))
17     text=arr[:]
18     arr.clear()
19     for i in text:
20         if i not in stopwords.words('english'):
21             arr.append(i)
22     return arr
```

'WINNER!! As a valued network customer you have been selected to receive a £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.'

- b. **Stemming:** It is a text normalization technique in natural language processing that involves reducing words to their base or root form, known as the "stem." This process aims to capture the core meaning of words by removing common suffixes or prefixes, facilitating the grouping of similar variations. For example, the word "running" stems from "run."

run

['i',
, 'me',
, 'my',
, 'myself',
, 'we',
, 'our',
, 'ourselves',
, 'you',
, 'you're',
, 'you've',
, 'you'll',
, 'you'd',
, 'your',
, 'yours',
, 'yourself',
, 'yourselves',
, 'he',
, 'him',
, 'his',
, 'himself',
, 'she',
, 'she's',
, 'her',
, 'hers',
, 'herself',
, 'it',
, 'it's',
, 'its',
, 'itself',
, 'they',
, 'them',
, 'their',
, 'theirs',
, 'themselves',
, 'what',
, 'which',
, 'who',
, 'whom',

- c. Removal of punctuation marks and retaining only the alphanumeric values.

After tokenization and stemming of the data, we converted all data into the same datatype of string to avoid any unnecessary delays and errors:

```

217] 1 preprocess(df['Text'][32])

['fear', 'faint', 'housework', 'quick', 'cuppa']

1 df['preprocessed_text']=df['Text'].apply(preprocess)
2 ...
3 We first apply 'type' function on the pre
4 processed text column to check the type, then use the 'astype'
5 to convert all data into same data type
6 ...
7 df['preprocessed_text']=df['preprocessed_text'].astype(str)

219] 1 df.head()

```

	Target	Text	num_char	num_words	num_sent	preprocessed_text
0	0	Go until jurong point	21	4	1	['go', 'jurong', 'point']
1	0	Ok lar... Joking wif u oni...	29	8	2	['ok', 'lar', 'joke', 'wif', 'u', 'oni']
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	['free', 'entri', '2', 'wkli', 'comp', 'win', ...]
3	0	U dun say so early hor... U c already then say...	49	13	1	['u', 'dun', 'say', 'earli', 'hor', 'u', 'c', ...]
4	0	Nah I don't think he goes to usf	32	9	1	['nah', 'think', 'goe', 'usf']

Building a Wordcloud to show the spread in ham and spam categories:

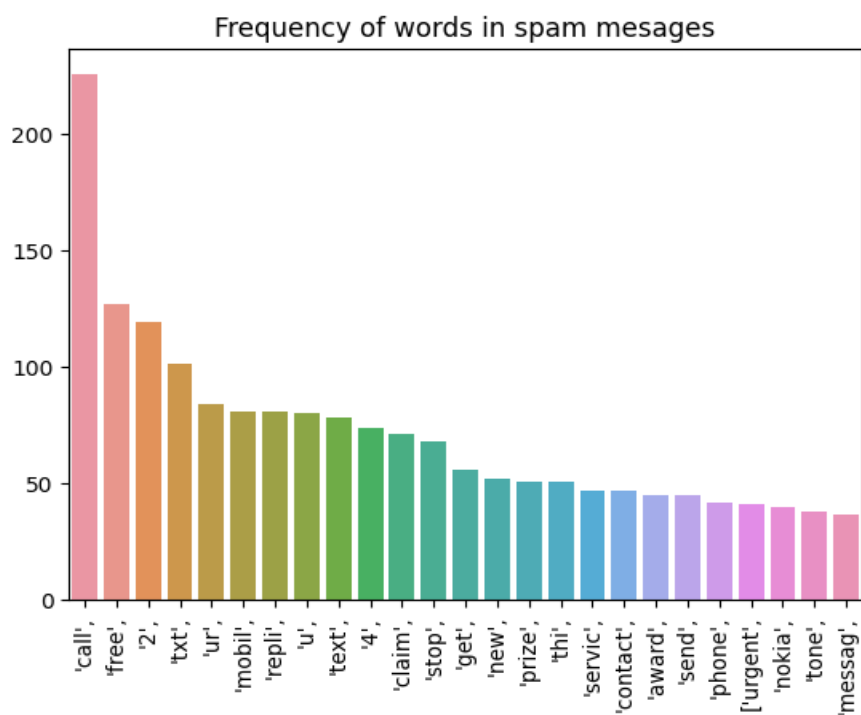
For better analysis and visualization of the distribution of the words, they provide a quick visual summary of the most frequently occurring words in a given text or dataset. The size of each word in the cloud is proportional to its frequency, making it easy to identify key terms and patterns.

```

1 !pip install wordcloud
2 from wordcloud import WordCloud
3 wc=WordCloud(width=500,height=500,max_font_size=100,max_words=100,background_color='white')
4 spamwc=wc.generate(df[df['Target']==1]['preprocessed_text'].str.cat(sep=" "))
5 plt.imshow(spamwc)#shows frequently occuring words in spam messages.

```


	0	1
0	'call',	225
1	'free',	127
2	'2',	119
3	'txt',	101
4	'ur',	84
5	'mobil',	81
6	'repli',	81
7	'u',	80
8	'text',	78
9	'4',	74
10	'claim',	71
11	'stop',	68
12	'get',	56
13	'new',	52
14	'prize',	51
15	'thi',	51
16	'servic',	47
17	'contact',	47
18	'award',	45
19	'send',	45
20	'phone',	42
21	['urgent',	41
22	'nokia',	40
23	'tone',	38
24	'messag',	37



The Frequency analysis of the most commonly used words in spam messages can be understood through the above bar plot. This can also help us to define the weights of a few commonly used words in our learning algorithm.

IMPORTING LIBRARIES FOR MACHINE LEARNING ALGORITHMS:

```
[ ] from sklearn.feature_extraction.text import CountVectorizer#using bag of words
```

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
```

```
▶ from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[ ] from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB#comparing multinomial nb with other classifiers
```

- a. **sklearn.feature_extraction:** From this library, we import the function `train_test_split`, which is used to split a dataset into training and test sets.
- b. **sklearn.metrics:** it includes functions for precision, recall, confusion matrix, etc., so according to our requirements, we import the functions of `accuracy_score`, `confusion_matrix`, and `precision_score`, which will be used to get the accuracy score, confusion matrix, and precision score of different ML algorithms.
- c. **sklearn.naive_bayes:** it includes the functions for all different versions that use Bayes logic. Here, we import Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes.
- d. **sklearn.ensemble:** it includes different ensemble classifiers; from this, we are importing the Random Forest classifier, Bagging Classifier, Extra Trees Classifier, AdaBoost Classifier, and Gradient Boosting Classifier.
- e. **sklearn.svm:** from it, we import `SVC`
- f. **sklearn.neighbors:** from it, we import `KNeighborsClassifier`

```
cv=CountVectorizer()

[ ] x=cv.fit_transform(df['preprocessed_text']).toarray()#toarray used to convert sparse array to dense array

[ ] y=df['Target'].values

[ ] from sklearn.model_selection import train_test_split

[ ] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=2)
```

- a. **CountVectorizer:** This is used to convert a collection of text documents to a matrix of token counts. Then we create an object of it and apply it to variable preprocessed text, and we dense it using toarray().
- b. **train_test_split:** this function splits the dataset into two groups, one for the training set and another for the test set. test_size=0.3 indicates the size of the test set, and random_state=2 is for making the test set random.

APPLYING AND EVALUATING DIFFERENT ML MODELS

Naïve Bayes :

This algorithm is from the family of “probabilistic classifiers.” Its application is inspired by the Bayes theorem. Despite its simplicity and the naive assumption of feature independence, Naive Bayes can be surprisingly effective, particularly in text classification tasks, due to its assumption of independence, capability to handle high-dimensional data, and various other reasons.

Here we are comparing different Naive Bayes models, namely Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes. We compare these models with both approaches (treating as bag-of-words and with tf-idf).

Gaussian Naive Bayes Classifier:

The Gaussian Naive Bayes Classifier is a variant of the Naive Bayes algorithm designed for datasets with continuous features. It assumes that the features within each class are normally (Gaussian) distributed.

```
[ ] gnb=GaussianNB()
```

it creates an instance of GaussianNB() and stores it in the variable gnb.

```
[ ] gnb.fit(x_train,y_train)
    y_pred1=gnb.predict(x_test)

[ ] from sklearn.metrics import accuracy_score,confusion_matrix,precision_score

[ ] print(accuracy_score(y_test,y_pred1))

0.8405081157374735

[ ] print(precision_score(y_test,y_pred1))#very low precision score

0.4230769230769231

[ ] print(confusion_matrix(y_test,y_pred1))

[[1037  210]
 [  16  154]]
```

gnb.fit(x_train,y_train): It trains the instance of the GaussianNB classifier (i.e., gnb) on the training set. Here x_train is the feature matrix of the training set, and y_train is the target variable of the training set.

gnb.predict(x_test): It makes predictions on the test data using the trained Gaussian Naive Bayes model.

print(accuracy_score(y_test,y_pred1)), print(precision_score(y_test,y_pred1)), print(confusion_matrix(y_test,y_pred1)): These lines calculate and print the accuracy, precision, and confusion matrix of the Gaussian Naive Bayes model on the test set.

Multinomial Naive Bayes Classifier:

Multinomial Naive Bayes is a variant of the Naive Bayes algorithm designed for text classification tasks with discrete features, such as word counts. It models the probability distribution of feature values (word frequencies) for each class

```
mnb=MultinomialNB()
```

it creates an instance of MultinomialNB() and stores it in the variable mnb.

```
[ ] mnbs.fit(x_train,y_train)
    y_pred2=mnbs.predict(x_test)

[ ] print(accuracy_score(y_test,y_pred2))

0.9703599153140438

▶ print(precision_score(y_test,y_pred2))

0.8636363636363636

[ ] print(confusion_matrix(y_test,y_pred2))

[[1223   24]
 [   18 152]]
```

mnbs.fit(x_train,y_train): It trains the instance of the MultinomialNB classifier (i.e. gnb) on the training set.

mnbs.predict(x_test): It makes predictions on the test data using the trained multinomial Naive Bayes model.

print(accuracy_score(y_test,y_pred1)), print(precision_score(y_test,y_pred1)), print(confusion_matrix(y_test,y_pred1)): These lines calculate and print the accuracy, precision, and confusion matrix of the multinomial Naive Bayes model on the test set.

Bernoulli Naive Bayes Classifier:

```
mnbs=BernoulliNB()
bnb=BernoulliNB()
```

It creates an instance of BernoulliNB() and stores it in the variable bnb.

```
[ ] bnb.fit(x_train,y_train)
    y_pred3=bnb.predict(x_test)
```

```
▶ print(accuracy_score(y_test,y_pred3))
```

```
👤 0.958362738179252
```

```
[ ] print(precision_score(y_test,y_pred3))#good precision---
```

```
0.9663865546218487
```

```
[ ] print(confusion_matrix(y_test,y_pred3))
```

```
[[1243    4]
 [  55   115]]
```

bnb.fit(x_train,y_train): It trains the instance of the BinomialNB classifier (i.e., gnb) on the training set.

bnb.predict(x_test): It makes predictions on the test data using the trained Binomial Naive Bayes model.

print(accuracy_score(y_test,y_pred1)), print(precision_score(y_test,y_pred1)), print(confusion_matrix(y_test,y_pred1)): These lines calculate and print the accuracy, precision, and confusion matrix of the Binomial Naive Bayes model on the test set.

Using TF-IDF:

The TF-IDF transforms a collection of preprocessed text data into numerical features


```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

[ ] tfidf=TfidfVectorizer()

[ ] x=tfidf.fit_transform(df['preprocessed_text']).toarray()
```

from sklearn.feature_extraction.text import TfidfVectorizer: imports the TfidfVectorizer class from scikit-learn, which is used to convert a collection of text documents into a matrix of TF-IDF features.

TfidfVectorizer(): it creates an instance of TfidfVectorizer

Now we apply this to preprocessed data.

```
[ ] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=2)
```

This separates the test and training data from the dataset after applying TfidfVectorizer to it.

Now again, all three models are trained and tested on it similarly as above, and the precision, accuracy, and confusion matrix are obtained.

Finally, we found that either Bernoulli Naive Bayes with bag-of-words(CountVectorizer) or Multinomial Naive Bayes with Tfidf gave better results, so we chose among them.

So, we use Multinomial Naive Bayes with Tfidf to compare with the rest of the classifiers in further proceedings of the program.

Code for getting the precision, accuracy, and confusion matrix for different models

```
[ ] svc=SVC(kernel='sigmoid',gamma=1.0)
    knc=KNeighborsClassifier()
    bc=BaggingClassifier(n_estimators=60,random_state=2)
    et=ExtraTreesClassifier(n_estimators=60,random_state=2)
    adc=AdaBoostClassifier(n_estimators=60,random_state=2)
    gdc=GradientBoostingClassifier(n_estimators=60,random_state=2)
    mnb=MultinomialNB()
    dtc=DecisionTreeClassifier(max_depth=5)
    lrc=LogisticRegression(solver='liblinear',penalty='l1')
    rfc=RandomForestClassifier(n_estimators=60,random_state=2)
```

svc=SVC(kernel = 'sigmoid',gamma=1.0): creates an instance of Support Vector Classifier using a sigmoid kernel and sets the gamma parameter to 1.0.

knc=KNeighborsClassifier(): creates an instance of K neighbor classifier

bc = BaggingClassifier(n_estimators=60,random_state=2): creates an instance, uses 60 decision trees, and sets random for reproducibility.

Similarly, instances of all other classifiers are made.

```
[ ] clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB' : mnb,
    'DT' : dtc,
    'LR' : lrc,
    'RF' : rfc,
    'AdaBoost' : adc,
    'BgC' : bc,
    'ETC' : et,
    'GBDT' : gdc,
}##preparing a dictionary of all the objects so that we can iterate over it using a function and apply all classifiers one by one
```

Here, a clfs dictionary is made where keys represent the names of different classifiers and values correspond to classifier instances. This is done so that we can easily iterate over each classifier using a single function.

```
[ ] def fit_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)
    cm=confusion_matrix(y_test,y_pred)

    return accuracy,precision,cm
```

clf.fit_classifier: is a function that takes in input five things, which are clf (i.e., the classifier model) and x_train (feature matrix of the training). sey_train (the target variable of the training set), x_test (the feature matrix of the test set), and y_test (the target variable of the test set)

clf.fit(x_train,y_train): trains the classifier model on the training set.

clf.predict(x_test): apply the classifier on the test set.

Then the accuracy, precision, and confusion matrix are computed, and the function returns those values.

```
1 accuracy_scores = []
2 precision_scores = []
3 confusion_scores=[]
4 for name,clf in clfs.items():
5
6     current_accuracy,current_precision,current_confusion = fit_classifier(clf, x_train,y_train,x_test,y_test)
7
8     print("For ",name)
9     print("Accuracy - ",current_accuracy)
10    print("Precision - ",current_precision)
11    print("Confusion matrix - ", current_confusion)
12
13    accuracy_scores.append(current_accuracy)
14    precision_scores.append(current_precision)
15    confusion_scores.append(current_confusion)
16    plt.figure(figsize=(8, 6))
17    sns.heatmap(current_confusion, annot=True, fmt="d", cmap="Blues",
18               xticklabels=clf.classes_, yticklabels=clf.classes_)
19    plt.title(f'Confusion Matrix for {name}')
20    plt.xlabel('Predicted')
21    plt.ylabel('Actual')
22    plt.show()
```

accuracy_scores, **precision_scores**, and **confusion_matrix** are three lists that are initially empty and will be used to store the corresponding values of different classifiers.

for name,clf in clf.items(): the code iterates over all items in clfs.

```
current_accuracy,current_precision,current_confusion = fit_classifier(clf, x_train,y_train,x_test,y_test)
```

This line iterates over the `fit_classifier()` function for all the classifiers and stores the results in three different variables. Then, the values are printed and appended to the corresponding list.

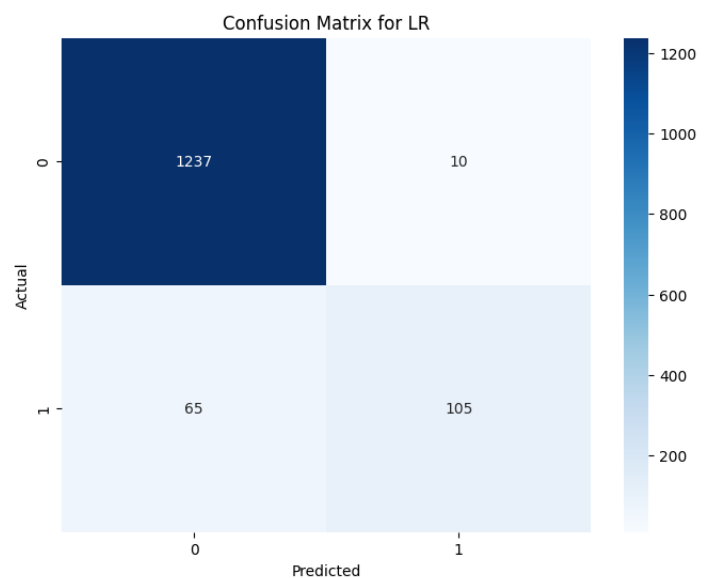
A. Logistic regression:

Logistic regression is a supervised classification algorithm. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered "1." Just like linear regression assumes that the data follows a linear function, logistic regression models the data using the sigmoid function.

The accuracy value, precision value, and confusion matrix using the Logistic Regression model:

Accuracy - 0.9470712773465068

Precision - 0.9130434782608695



B. Decision Tree Classifier:

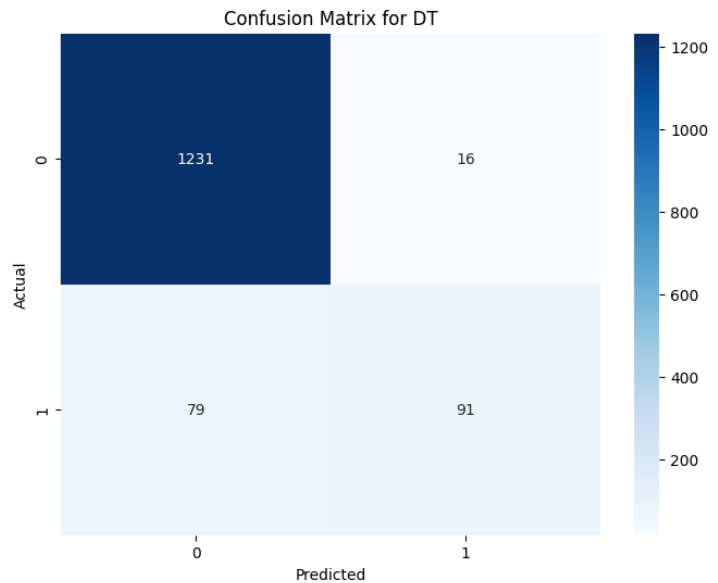
It is a supervised learning technique. A decision tree follows a tree structure and checks if an attribute or a set of attributes satisfies a condition, and based on the result of the check, subsequent checks are performed. The tree splits the data into different parts based on these checks. The nodes at different levels are selected using different techniques like entropy, Chi-square, and Gini Index. Its major drawback is that it is computationally expensive.

The accuracy value, precision value, and confusion matrix using the Decision Tree Classifier model:

For DT:

Accuracy - 0.9329569513055752

Precision - 0.8504672897196262



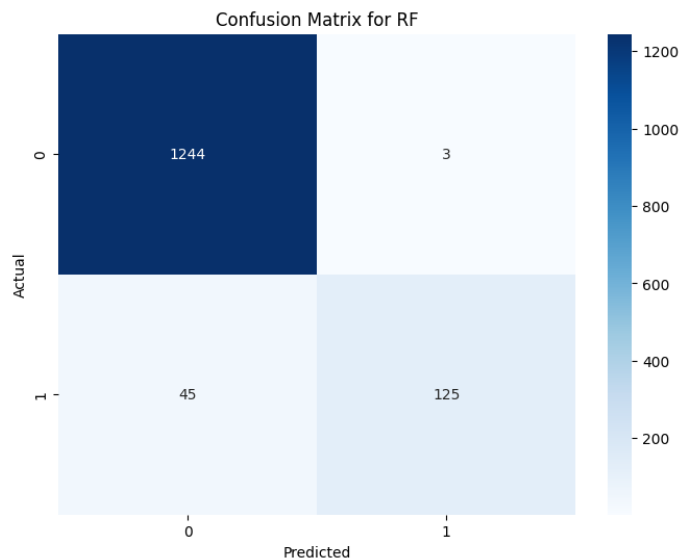
C. Random Forest:

Random forests are an ensemble learning technique that is built using decision trees. Random forests involve creating multiple decision trees using bootstrapped datasets of the original data and randomly selecting a subset of variables at each step of the decision tree. The model then selects the mode of all of the predictions in each decision tree. By relying on a “majority wins” model, it reduces the risk of error from an individual tree. Random Forest can easily handle noisy data. Also, to avoid overfitting, Random Forest is really good. This makes our result more generalized and gives us a better prediction.

The accuracy value, precision value, and confusion matrix using the Random Forest model are

Accuracy - 0.9661256175017643

Precision - 0.9765625



D. Support Vector Classifier(SVC):

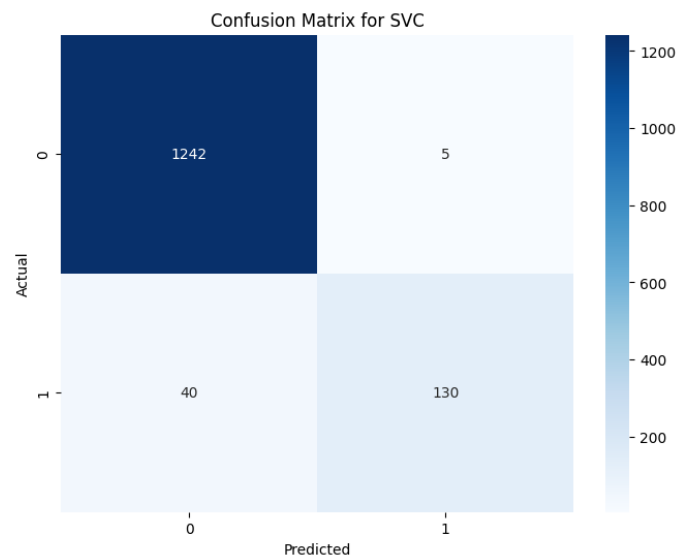
Support Vector Machine, or SVM, is one of the most popular Supervised Learning algorithms, which is used for classification as well as regression problems. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed a Support Vector Machine (SVM).

SVM is capable of doing both classification and regression. The benefit is that you can capture much more complex relationships between your data points without performing difficult transformations on your own.

The accuracy value, precision value, and confusion matrix using the Random Forest model:

Accuracy - 0.9682427664079041

Precision - 0.9629629629629629

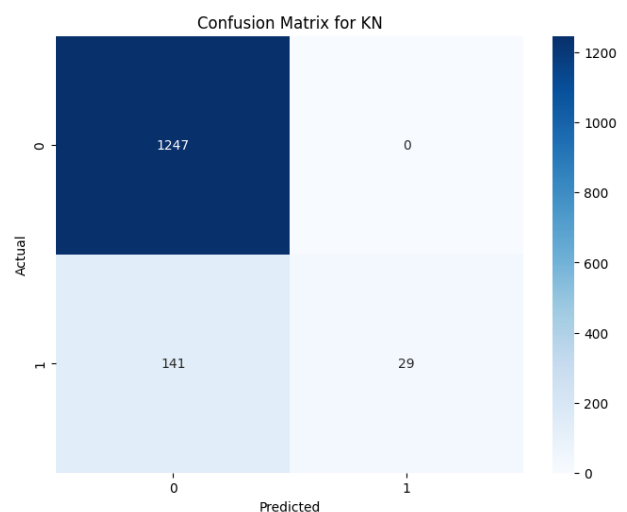


E. K-Nearest

Neighbours(KNN):

It is another supervised learning algorithm that comes under the "similarity-based model" and follows weak learning. It selects k neighbors based on the distance from them and assigns that class to the test pattern, which is most common for these k neighbors.

The accuracy value, precision value, and confusion matrix using the KNN model:



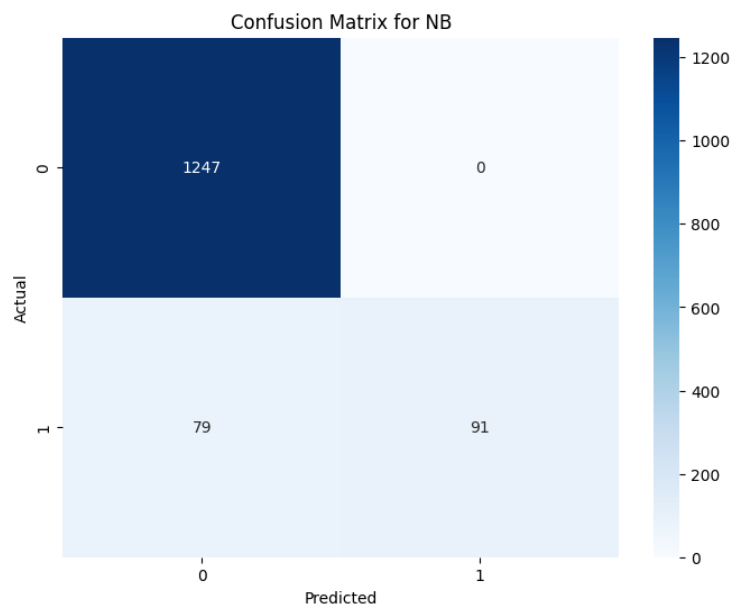
Accuracy - 0.9004940014114327
Precision - 1.0

F. Naive Bayes:

This algorithm is from the “probability-based” model of supervised learning and follows Naive Bayes. Here, we are using multinomial Naive Bayes with tf-idf as it works well on textual data due to its assumption of independence and other factors.

The accuracy value, precision value, and confusion matrix using the logistic regression model:

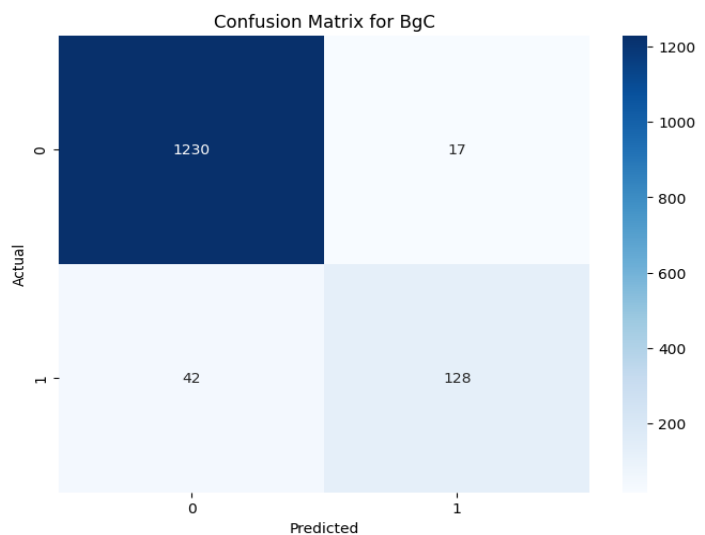
Accuracy - 0.9442484121383204
Precision - 1.0



G. Bagging Classifier:

A bagging classifier is an ensemble machine learning algorithm that combines the predictions of multiple base classifiers, typically decision trees.

The accuracy value, precision value, and confusion matrix using the Bagging classifier model:



For BgC

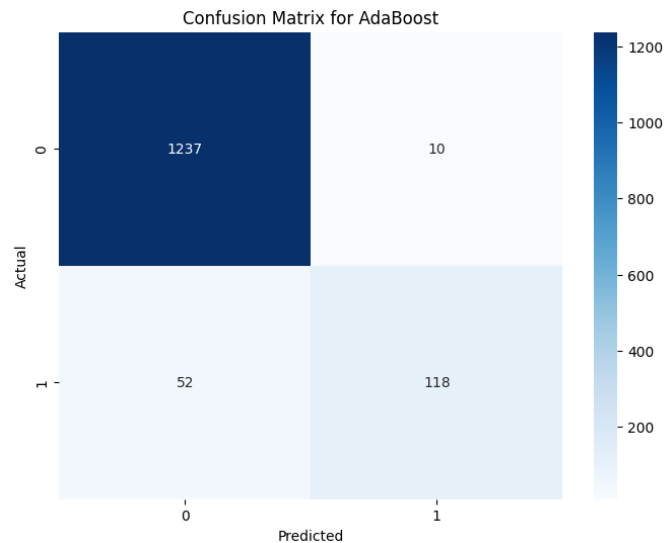
Accuracy - 0.958362738179252

Precision - 0.8827586206896552

H. AdaBoost Classifier:

AdaBoost (Adaptive Boosting) Classifier is an ensemble learning algorithm that combines the predictions of weak learners (typically shallow decision trees) to create a strong classifier. It assigns weights to instances in the training data, focusing more on misclassified instances in subsequent iterations.

The accuracy value, precision value, and confusion matrix using the AdaBoost classifier model:

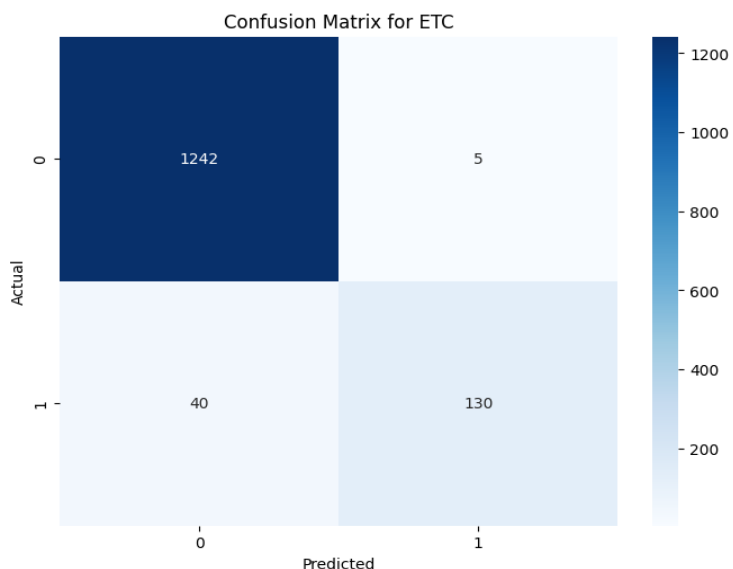


Accuracy - 0.9562455892731122

Precision - 0.921875

I. Extra Trees Classifier:

The Extra Trees Classifier is an ensemble learning algorithm that builds multiple decision trees on random subsets of the training data and features. It introduces additional randomness by selecting random thresholds for each feature at each node.



The accuracy value, precision value, and confusion matrix using the Extra Trees classifier model are

Accuracy - 0.9682427664079041

Precision - 0.9629629629629629

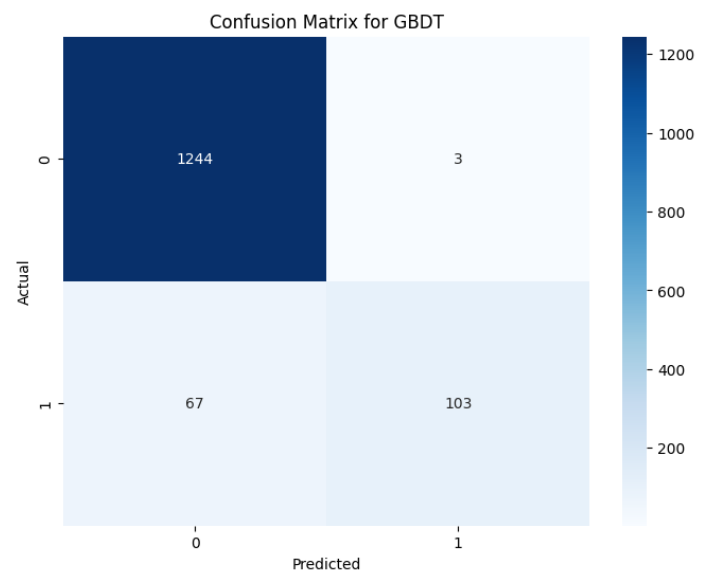
J. Gradient Boosting Classifier:

Gradient Boosting Classifier is an ensemble learning technique that builds a series of weak learners (usually decision trees) sequentially, with each new learner focusing on the mistakes of the previous ones. It minimizes errors by adjusting weights for each instance and combining the predictions of individual learners through a weighted sum.

The accuracy value, precision value, and confusion matrix using the Extra Trees classifier model:

Accuracy - 0.9505998588567396

Precision - 0.971698113207547



Name of the Model	Logistic Regression	Decision Tree	Random Forest	Support Vector	KNN	Naive Bayes	Bagging Classifier	AdaBoost classifier	Extra Tree Classifier	Gradient Boosting Classifier
Precision	0.9130434782608695	0.8504672897196262	0.9765625	0.9629629629629629	1.0	1.0	0.8827586206896552	0.921875	0.9629629629629629	0.97169813207547
Accuracy	0.9470712773465068	0.9329569513055752	0.9661256175017643	0.9682427664079041	0.9004940014114327	0.9442484121383204	0.958362738179252	0.9562455892731122	0.9682427664079041	0.9505998588567396

SELECTING THE MODEL

Here, we compare the accuracy and precision of different models.
The above code plots accuracy and precision for different models:

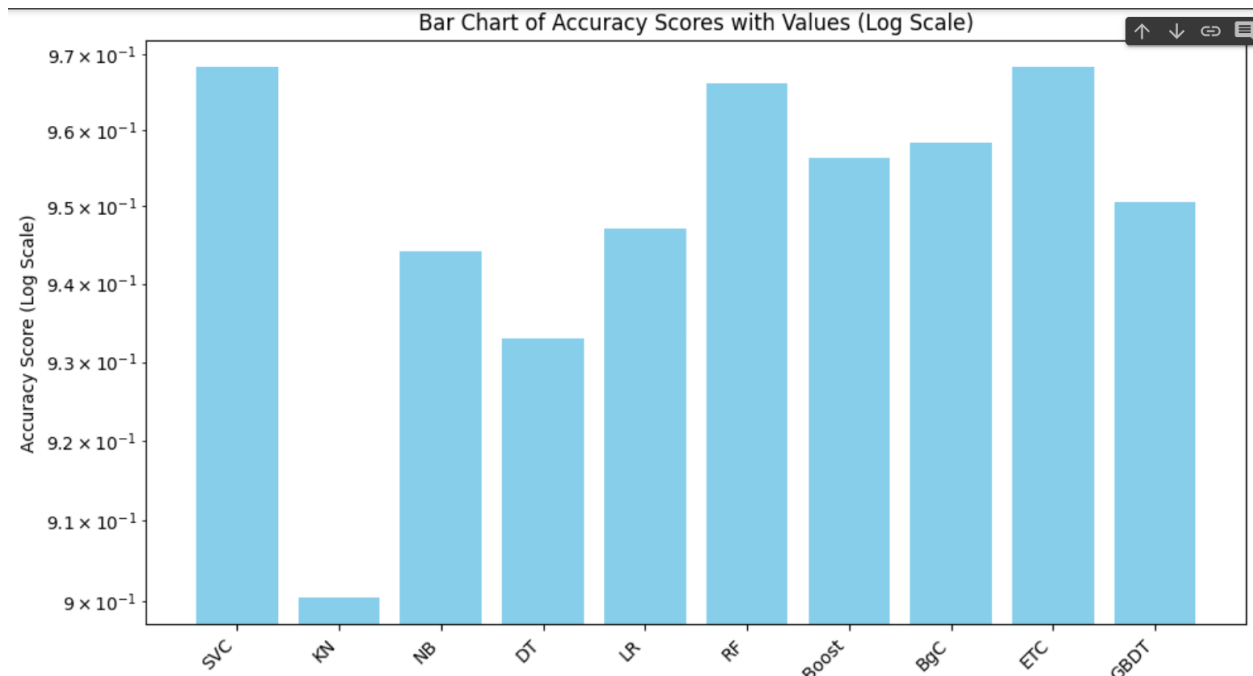
```
# Extract classifier names and corresponding scores
classifier_names = list(clfs.keys())
accuracy_scores = accuracy_scores

# Create bar chart for accuracy scores with values inside bars
plt.figure(figsize=(10, 6))
plt.yscale('log')
bars = plt.bar(classifier_names, accuracy_scores, color='skyblue')

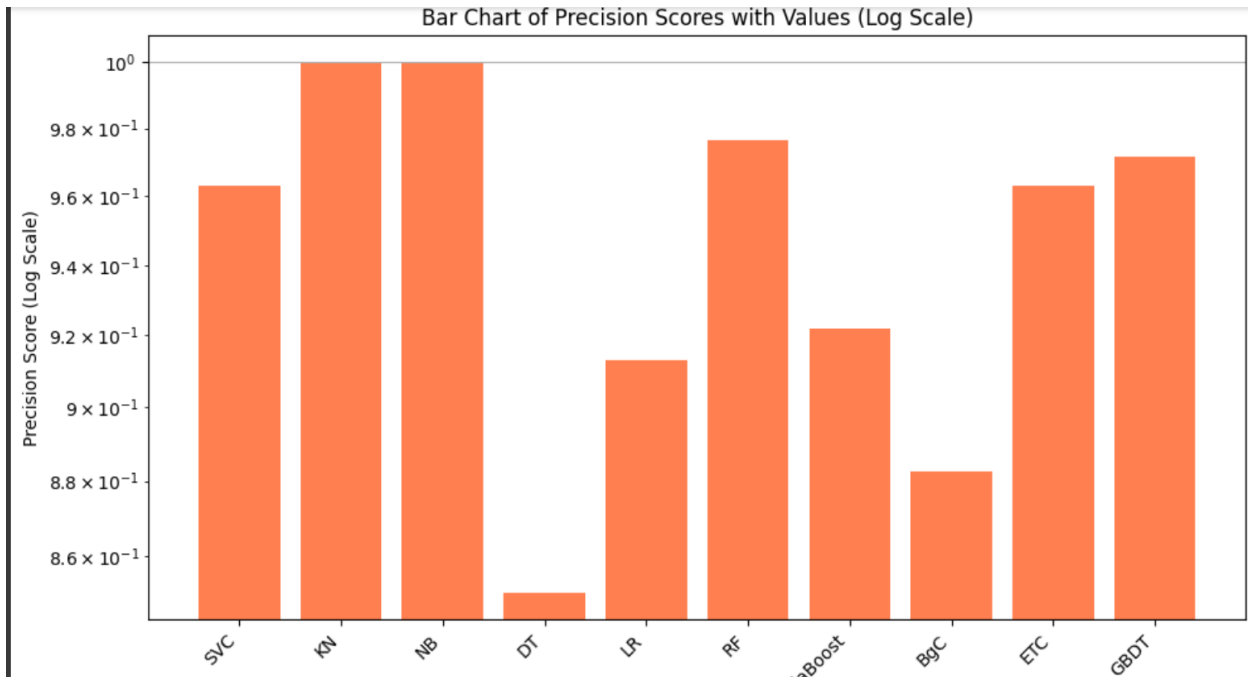
# Add value labels to each bar using annotate() function
for bar, accuracy in zip(bars, accuracy_scores):
    yval = bar.get_height() / 2
    plt.annotate(round(accuracy, 4), (bar.get_x() + bar.get_width() / 2, yval),
                 xytext=(0, -5), textcoords='offset points', ha='center', va='center', fontsize=8)

plt.xlabel('Classifier')
plt.ylabel('Accuracy Score (Log Scale)')
plt.title('Bar Chart of Accuracy Scores with Values (Log Scale)')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

The Bar Chart of Accuracy Scores:



The precision graph for precision is as follows:



Based on these graphs, we can use the model with the most accuracy or precision according to the requirements.

But if we have to choose one considering both factors, then the **Random Forest Classifier** is appropriate as it gives reasonable accuracy and precision.

INFERENCES:

- In the data taken from the repository, the normal messages were labeled as ham, whereas the spam messages were labeled as spam.
- Nearly 88% of messages were normal messages, whereas 12% were spam, which implies that the probability of getting a spam message is less than 0.15.
- The average word count in a text message is 15, and the number of lines is approximately 2 for any text message sent between the end users.
- The spam messages have an average word count of 23 words per message, which is greater than the average number of words in ham messages, which is 14 words. We can infer here that spam messages tend to be longer in terms of word count, so the higher the word count, the higher the probability of it being a spam message.
- The exceptional outliers of ham messages in terms of word count also help us to infer that after a threshold, a very long message will not be a spam message for sure, as was quite evident from the box plot that we made.
- Words like 'free', 'claim', 'cash', "reply", and 'call' are common in spam messages,, it helps in deciding the weights of the words to get the prediction properly.
- Different classification models give different values of accuracy and precision to make better predictions with many features, i.e., the word count, character count, and message itself.
- Among Naive Bayes classifiers, Multinomial Naive Bayes with Tfidf gives the best result.
- All the different classifiers (like Naive Bayes, Random Forest, etc.) have different advantages and limitations.
- SVC gives the highest accuracy, whereas KNN gives high precision, so based on the requirements, one needs to choose a suitable model.
- Before training the model, we applied various preprocessing techniques to clean the data through stemming, tokenization, datatype conversion, punctuation removal, and sorting the overall raw dataset into a list of messages, which is itself a list of the core words of that text message.

QUESTIONS WE CAN NOW ANSWER:

- Can we accurately predict whether the messages are spam based on other attributes?
- Is there a relationship between words used in a message and it being spam?
- Is there a relationship between the message being spam and the length of the message?
- Can we identify any trends or patterns in the data that may be useful for predicting whether the messages are spam?
- How accurately can we predict the messages to be spam or not using different machine learning algorithms, and which algorithm performs the best on this dataset?

CONCLUSIONS:

- The support Vector Classifier model had the highest accuracy among the models that were evaluated for the SMS Spam Detection dataset.
- Naive Bayes and KNeighbor Classifier have the highest precision among the models evaluated for the SMS Spam Detection dataset.
- The decision tree gives the best result when we want a fair tradeoff between accuracy and precision. This suggests that decision trees can be a good model to categorize messages as spam or not.
- Overall, our analysis has helped us to understand the SMS Spam Detection dataset and how, based on the features of the messages, we can predict whether they are spam or not.
- In summary, the SMS Spam Detection dataset provides a rich source of information about the features of spam and normal messages.