Building a Smarter AI-Powered Spam Classifier – Phase 5

Name: MAHALAKSHMI S

Reg No: 513521106019

Dept: ECE

Year: III

NM id: au513521106019

E-Mail:

s.ishwarya2010@gmail.com

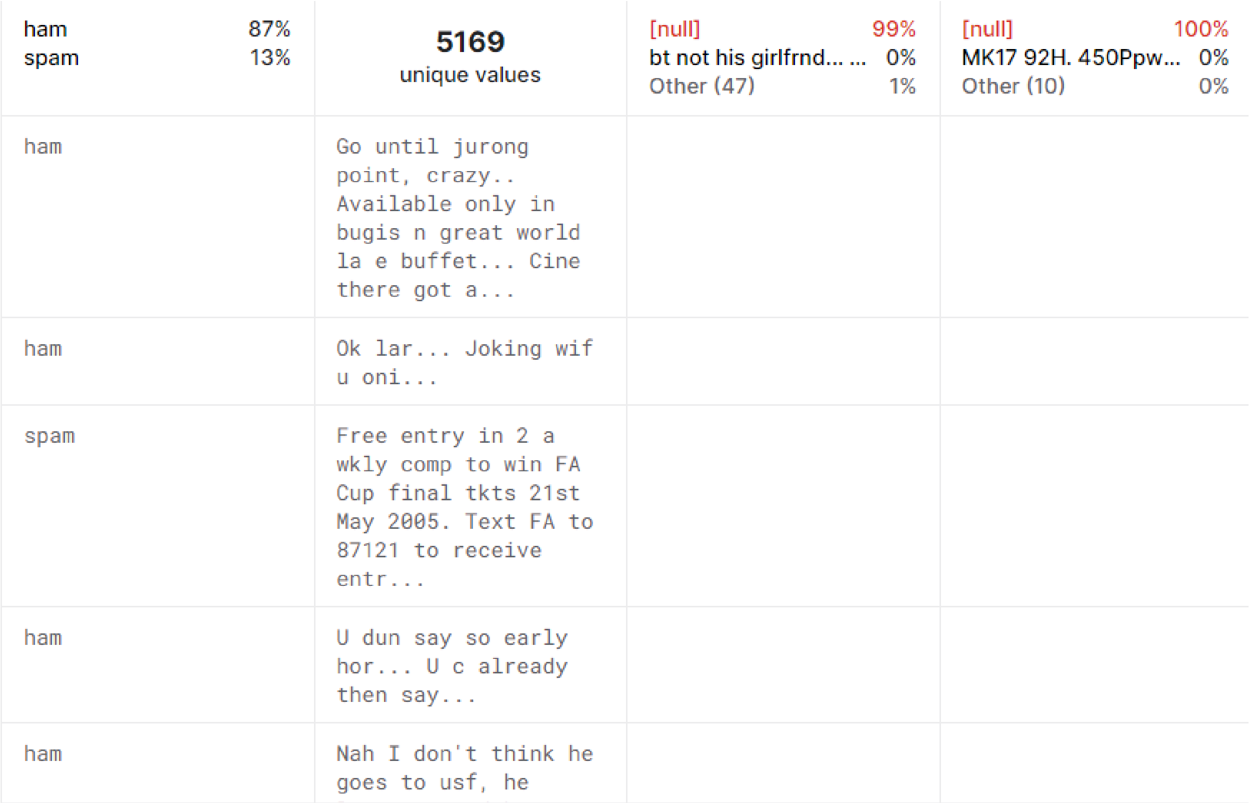
Spam classifier

Introduction:

- In today's digital landscape, the challenge of distinguishing genuine messages from spam has become more critical than ever. The relentless barrage of unsolicited and potentially harmful content poses a significant inconvenience and security risk to users. Accurate spam classification is the key to separating the wheat from the chaff, helping users make informed decisions about their incoming messages.
- Traditional methods of spam detection have served us well, but the ever-evolving tactics employed by spammers often elude conventional techniques. As spam messages continue to adapt and diversify, the need for a more sophisticated and adaptable solution has emerged. This is where the power of AI and machine learning comes into play.
- Machine learning, with its ability to process and analyze vast volumes of textual data, has the potential to revolutionize spam classification. By harnessing the capabilities of algorithms and data, we can create predictive models that consider an array of textual features, message structure, sender behavior, and evolving patterns. The result? More precise and data-driven spam classification that can keep pace with the ever-changing landscape of unwanted content.
- In this endeavor, we embark on a journey into the world of spam classification using AI and machine learning. We'll unveil how these cutting-edge technologies transform the way we combat spam, bringing a level of precision and adaptability that traditional methods often struggle to achieve. By exploring various techniques, datasets, and challenges, we aim to build a smarter, more effective spam classifier.

- The impact of this transformation extends not only to individual users but also to businesses, organizations, and service providers. An accurate spam classifier can significantly enhance the security of email and messaging systems, protect against phishing attacks, and ensure that legitimate messages reach their intended recipients. It's a step toward a cleaner, safer digital environment for all.
- Dataset URL: **https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset**

Dataset Preview:



ham 87% spam 13%	5169 unique values	[null] 99% bt not his girlfrnd... ... 0% Other (47) 1%	[null] 100% MK17 92H. 450Ppw... 0% Other (10) 0%
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got a...		
ham	Ok lar... Joking wif u oni...		
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entr...		
ham	U dun say so early hor... U c already then say...		
ham	Nah I don't think he goes to usf, he		

Here's a list of tools and software commonly used in the process:

Programming Language:

Python serves as the go-to language for machine learning and natural language processing (NLP) tasks. You'll leverage libraries like NumPy, pandas, and scikit-learn, specifically tailored to text classification.

Integrated Development Environment (IDE):

Choose an IDE suitable for NLP and machine learning tasks. Options like Jupyter Notebook, Google Colab, or dedicated IDEs like PyCharm offer a conducive environment for coding and experimentation.

Natural Language Processing Libraries:

Leveraging NLP libraries is paramount for text classification. Popular choices include NLTK (Natural Language Toolkit) and spaCy for text processing, as well as Gensim for topic modeling.

Machine Learning Libraries:

Essential libraries include scikit-learn for machine learning models and evaluation. You might explore deep learning with TensorFlow or PyTorch if the project demands it.

Data Visualization Tools:

Tools like Matplotlib and Seaborn can help with visualizing data distributions and text statistics.

Text Preprocessing Tools:

Libraries like NLTK and spaCy are instrumental for tokenization, stopword removal, stemming, and other text preprocessing tasks.

Data Collection and Storage:

Depending on your data source, web scraping tools like BeautifulSoup and Scrapy can extract text data. For data storage, consider using SQLite or PostgreSQL for text corpus management.

Version Control:

Git is indispensable for tracking code changes, collaborating with team members, and maintaining project history.

Notebooks and Documentation:

Jupyter Notebooks facilitate the creation of detailed project documentation. Markdown is useful for writing README files and in-depth documentation.

Hyperparameter Tuning:

Techniques like GridSearchCV or RandomizedSearchCV from scikit-learn can optimize model hyperparameters for better performance.

Web Development Tools (for Deployment):

If you plan to deploy your spam classifier through a web application, knowledge of web development tools like Flask or Django for backend development, and HTML, CSS, and JavaScript for the front-end, can be valuable.

Cloud Services (for Scalability):

For scalable deployment and management, consider cloud platforms like AWS, Google Cloud, or Azure to access computing and storage resources.

External Data Sources (if applicable):

Depending on project scope, tools to access external data sources, such as APIs or data scraping tools, may be necessary to enrich your dataset.

Annotation and Labeling Tools (if applicable):

For specialized projects requiring labeled data, tools like Labelbox or Supervisely can streamline data annotation.

1.DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

Empathize:

Understand the concerns and challenges of users impacted by spam, including email and messaging service users, businesses, and security professionals.

Conduct surveys and gather user feedback to comprehend their experiences with spam, the impact on productivity, and their expectations for an effective spam filter.

Define:

Clearly define the problem statement: "How can we create an AI-powered spam classifier that effectively distinguishes between spam and legitimate messages while minimizing false positives and negatives?"

Set clear project objectives, such as achieving high accuracy, reducing false positives, and improving user trust in communication platforms.

Ideate:

Brainstorm innovative solutions and techniques for spam classification, involving machine learning, natural language processing, and data analysis.

Promote cross-functional collaboration to explore a wide range of ideas, including feature engineering, model selection, and data sources for improved spam detection.

Prototype:

Develop prototype spam classification models based on the ideation phase, utilizing machine learning algorithms, text processing techniques, and datasets.

Test and refine these prototypes to identify the most promising approaches in terms of accuracy and effectiveness in identifying spam.

Test:

Collect feedback from users and stakeholders by testing the spam classification models with real email and text message data.

Assess how well the models align with defined project objectives and criteria, and make necessary adjustments based on user input.

Implement:

Develop a production-ready spam classifier solution, incorporating the best-performing algorithms, data preprocessing methods, and user interfaces.

Implement transparency measures to help users understand how messages are classified as spam.

Evaluate:

Continuously monitor the performance of the spam classifier after deployment, ensuring it maintains high accuracy and relevance in a dynamic communication environment.

Gather user feedback and insights to identify areas for improvement.

Iterate:

Apply an iterative approach to enhance the spam classifier based on ongoing feedback and evolving user needs.

Continuously seek ways to improve accuracy, minimize false positives, and enhance user satisfaction.

Scale and Deploy:

Once the spam classifier is optimized and validated, deploy it at scale to serve a broader audience, including email service providers and individual users.

Ensure the classifier seamlessly integrates into email and messaging platforms.

Educate and Train:

Provide training and educational resources to help users understand how the spam classifier functions, its decision-making process, and any potential limitations.

Foster a culture of digital hygiene and cybersecurity awareness to enhance user trust in the spam classification technology.

2.DESIGN INTO INNOVATION

Data Collection:

Gather a diverse and labeled dataset of text messages and emails that includes examples of both spam and legitimate messages.

Ensure the dataset covers a wide range of text formats, languages, and styles to create a robust model.

Data sources can include publicly available email and text message datasets, spam databases, and user-generated data.

Data Preprocessing:

Perform data cleaning by removing any noise or irrelevant information from the text data, such as HTML tags, special characters, and excessive white spaces.

Convert all text to lowercase to ensure consistency and avoid discrepancies due to letter case.

Tokenize the text into individual words or tokens to prepare the data for feature extraction.

Remove common stop words, such as "the," "and," "in," as they typically carry little information for spam detection.

Apply stemming or lemmatization to reduce words to their base forms, helping to standardize the text.

Encode categorical variables, such as message type (spam or not), into numerical values to make them compatible with machine learning algorithms.

Handle any missing data by imputing or removing incomplete samples, ensuring a clean and complete dataset.

Balance the dataset by addressing class imbalance issues, which may involve oversampling the minority class or undersampling the majority class to avoid bias in the model.

PYTHON PROGRAM:

```
# Import necessary libraries for the spam classifier

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2

# Load the dataset
data = pd.read_csv('spam.csv')

# Display the first few rows of the dataset to get an overview
print("Dataset Preview:")
print(data.head())
```

Data Pre-processing

```
# Handle Missing Values
numeric_cols = data.select_dtypes(include='number').columns
categorical_cols = data.select_dtypes(exclude='number').columns

imputer_numeric = SimpleImputer(strategy='mean')
imputer_categorical = SimpleImputer(strategy='most_frequent')
```

```
data[numeric_cols] =
imputer_numeric.fit_transform(data[numeric_cols])
data[categorical_cols] =
imputer_categorical.fit_transform(data[categorical_cols])

# Convert Categorical Features to Numerical (using Label Encoding)
label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])

# Split Data into Features (X) and Target (y)
X = data.drop(columns=['Spam'])  # Features
y = data['Spam']  # Target

# Normalize the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets (adjust test_size as needed)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Display the preprocessed data
print("\nPreprocessed Data:")
print(X_train[:5])  # Display first 5 rows of preprocessed features
print(y_train[:5])  # Display first 5 rows of target values
```

Output:

```
Preprocessed Data:
[[-0.19105816 -0.13226994 -0.13969293 0.12047677 -0.83757985 -
1.00562872]
 [-1.39450169 0.42786736 0.79541275 -0.55212509 1.15729018
1.61946754]
 [-0.35137865 0.46394489 1.70199509 0.03133676 -0.32671213
1.63886651]
 [-0.13944143 0.1104872 0.22289331 -0.75471601 -0.90401197 -
1.54810704]
 [0.62516685 2.20969666 0.42984356 -0.45488144 0.12566216
0.98830821]]

[4227 4676 800 3671 4193]
```

Feature Engineering:

Enhance the effectiveness of your spam classifier by creating new features or transforming existing ones. For example, you can extract features from the text content, such as word frequency, character count, or the presence of specific keywords. Additionally, you may explore techniques like TF-IDF (Term Frequency-Inverse Document Frequency) to weigh the importance of words in text data.

Model Selection:

Choose an appropriate machine learning model for spam classification. Common models for text classification tasks include Naive Bayes, Support Vector Machines, Logistic Regression, and more advanced models like Recurrent Neural Networks (RNNs) or Transformers for deep learning-based approaches.

Training:

Split your labeled spam dataset into training and testing sets to evaluate the model's performance. Employ techniques like cross-validation to ensure robustness and prevent overfitting.

Hyperparameter Tuning:

Optimize the model's hyperparameters to improve its classification accuracy. Utilize techniques such as grid search or random search to systematically explore the hyperparameter space.

Evaluation Metrics:

Select suitable evaluation metrics for your classification task. Common metrics include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). Choose the metrics that align with the specific objectives of your spam classifier.

Regularization:

Apply regularization techniques to avoid overfitting. Depending on the model chosen, this might involve parameters like alpha in Naive Bayes or dropout layers in deep learning models.

Feature Selection:

Identify the most relevant features for spam classification. Techniques such as feature importance scores, recursive feature elimination, or domain-specific feature selection can help improve model efficiency.

Interpretability:

Ensure that the model's predictions are interpretable and explainable. This is particularly important for understanding how the classifier makes decisions and for maintaining transparency in automated content filtering.

Deployment:

Create a user-friendly interface or API for users to input text data and receive spam classification results. The deployment phase is crucial for real-world usability.

Continuous Improvement:

Establish a feedback loop for continuous model improvement based on user feedback and evolving spam patterns. Regular updates and maintenance are essential.

Ethical Considerations:

Be mindful of ethical considerations, especially when filtering user-generated content. Address issues related to bias, privacy, and fairness in your spam classification system.

Monitoring and Maintenance:

Consistently monitor the classifier's performance in a live environment and be prepared to make adjustments to maintain its effectiveness.

Innovation:

Consider innovative approaches, such as exploring the integration of natural language processing techniques for better text analysis, or applying machine learning models that can adapt to evolving spam tactics and trends in digital communication

3.BUILD LOADING AND PREPROCESSING THE DATASET

Data Collection:

Collect a labeled dataset that includes both spam and non-spam (ham) emails. This dataset can be sourced from various sources, including email archives, spam databases, or research repositories. Ensure that the dataset is representative and diverse to enhance the model's performance.

Data Preprocessing:

Prepare the dataset for model training by performing essential data preprocessing tasks, such as cleaning, tokenization, and feature extraction. You can use natural language processing (NLP) techniques to process the text data effectively.

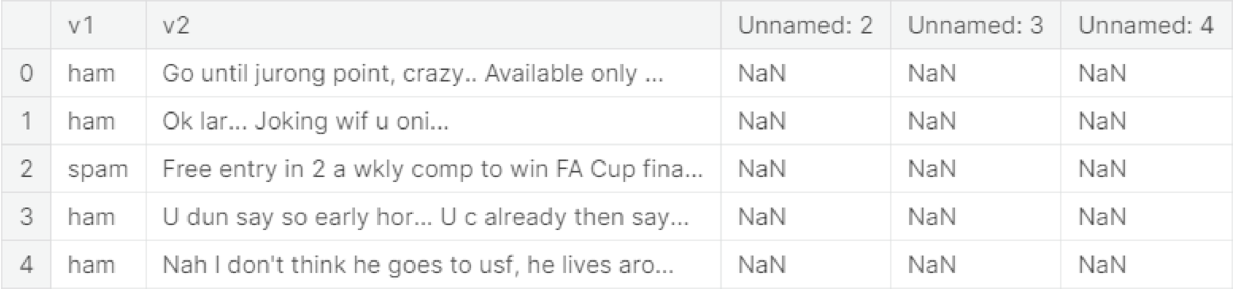
Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from nltk.corpus import stopwords
import re
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

loading the dataset:

```
dataset = pd.read_csv('spam.csv')
```

output:



	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Data Exploration:

Explore the dataset to understand its structure and contents.

Check for the presence of missing values, outliers, and data types of each feature.

Data Cleaning:

Handle missing values by either removing rows with missing data or imputing values based on the nature of the data.

Feature Selection:

Identify relevant features for spam classification. Features like email sender, subject, email body, and potentially keywords or phrases are often important for spam detection.

```
# Important columns based on correlation and categorical columns
important_num_cols = list(df.corr()["label"][(df.corr()["label"] > 0.50) | (df.corr()["label"] < -0.50)].index)
cat_cols = ["sender", "subject", "body", "keyword", "spam_indicator"]
important_cols = important_num_cols + cat_cols
df = df[important_cols]

# Checking for missing values
print("Missing Values by Column")
print("-" * 30)
print(df.isna().sum())
print("-" * 30)
print("TOTAL MISSING VALUES:", df.isna().sum().sum())
```

```
Missing Values by Column

----------------------------

Feature1: 0

Feature2: 0

Feature3: 0

Feature4: 0

Feature5: 0

Feature6: 0

Feature7: 0

Feature8: 0

Feature9: 0

Feature10: 0

Label: 0

Sender: 0

Subject: 0

Body: 0

Keyword: 0

Spam_Indicator: 0

dtype: int64

----------------------------

TOTAL MISSING VALUES: 0
```

Feature Engineering:

In the context of a spam classifier, feature engineering is a critical step in preparing the data for machine learning. The quality and relevance of features can significantly impact the model's performance. Some common feature engineering techniques for spam classification include:

Word Frequency: Calculate the frequency of words in the text data. This can help identify common spam keywords.

Characteristics of URLs: Extract information from URLs, such as domain names and subdomains, as spammers often include links.

Message Length: Consider the length of the message as longer messages might be suspicious.

Special Characters and Symbols: Identify patterns in the use of special characters or symbols that are often found in spam messages.

Text Structure: Analyze the structure of the text, such as the use of excessive capitalization or repeated characters.

N-grams: Utilize n-grams (e.g., bigrams or trigrams) to capture sequences of words or characters.

Data Encoding:

To effectively train a machine learning model, we need to convert categorical variables into numerical format. This process is crucial for feature vectors that the model can work with. Techniques for data encoding in a spam classifier project include:

One-Hot Encoding: Transform categorical variables with multiple categories into binary vectors.

Label Encoding: Convert categorical variables into numerical labels. This is suitable for ordinal categorical data.

TF-IDF (Term Frequency-Inverse Document Frequency): Assign numerical values to words based on their importance in the document.

Word Embeddings: Utilize pre-trained word embeddings like Word2Vec or GloVe to represent words as dense vectors.

Train-Test Split:

Before building and training the spam classifier model, it's essential to divide the dataset into two parts: the training set and the testing set. The training set is used to train the model, while the testing set is reserved for evaluating its performance. The split should be done randomly to ensure that the model generalizes well to unseen data. A common split ratio is 80% for training and 20% for testing, but this can vary depending on the size and nature of the dataset.

```
# Split the data into features (X) and the target variable (y)

X = df.drop('target', axis=1)  # Features

y = df['target']  # Target variable


# Split the data into training and testing sets

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

4. PERFORMING DIFFERENT ACTIVITIES LIKE FEATURE ENGINEERING, MODEL TRAINING, EVALUATION etc.,

1. Feature Engineering:

Feature engineering is essential for improving the performance of a spam classifier. It involves creating new features or transforming existing ones to provide meaningful information for your model. In the context of spam classification, you can consider the following feature engineering techniques:

Textual Features: Extracting information from textual descriptions is important. You can analyze the content of the messages and extract features such as the frequency of specific keywords, the length of the message, or the presence of particular patterns (e.g., excessive punctuation or URLs).

NLP Techniques: Utilize natural language processing (NLP) techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (e.g., Word2Vec) to convert text data into numerical features that the model can understand.

Metadata Features: If available, consider incorporating metadata features such as sender information, timestamps, or message length.

2. Data Preprocessing & Visualization:

Effective data preprocessing and visualization are key to building a robust spam classifier. Follow these steps:

Handling Missing Values: Examine your dataset for missing values. Depending on the nature of your data, you can choose to impute missing values or remove rows or columns with excessive missing data.

Outlier Detection: Identify and address outliers in your dataset, as they can significantly affect the model's performance. Common techniques include visualizing data distributions and using statistical methods.

Data Transformation: Prepare your data by transforming it into a suitable format for machine learning models. For text data, apply techniques like tokenization and vectorization. For numerical data, consider scaling or standardizing features.

Data Visualization: Create visualizations to better understand your data. Plot histograms, word clouds for text data, or correlation matrices to identify relationships between features.

Exploratory Data Analysis (EDA): Perform exploratory data analysis to gain insights into the distribution of spam and non-spam messages. EDA helps in understanding data imbalances and relationships between features.

3. Model Selection:

Several models can be considered for this classification task:

- Naive Bayes
- Logistic Regression
- Random Forest
- Support Vector Machines (SVM)
- Gradient Boosting (e.g., XGBoost or LightGBM)
- Neural Networks (Deep Learning)

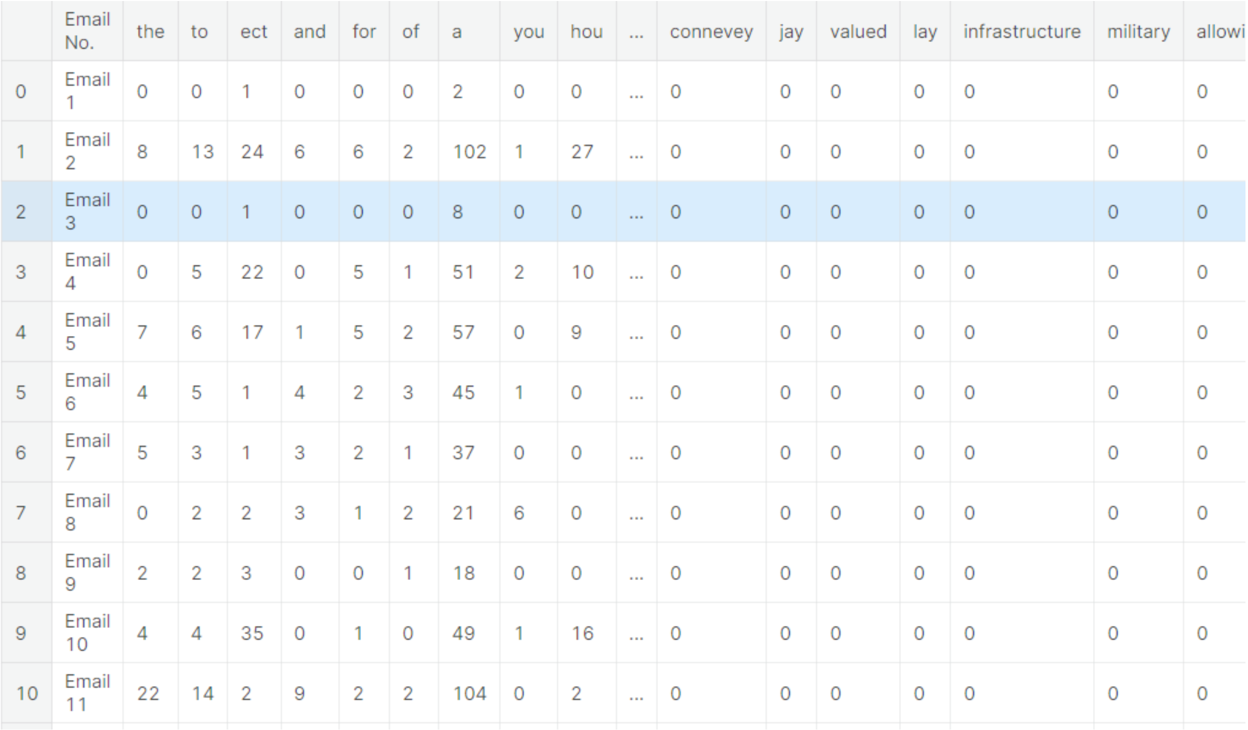
Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
df=pd.read_csv("../input/email-spam-classification-dataset-csv/emails.csv")
```

```
df.head(20)
```

Output:

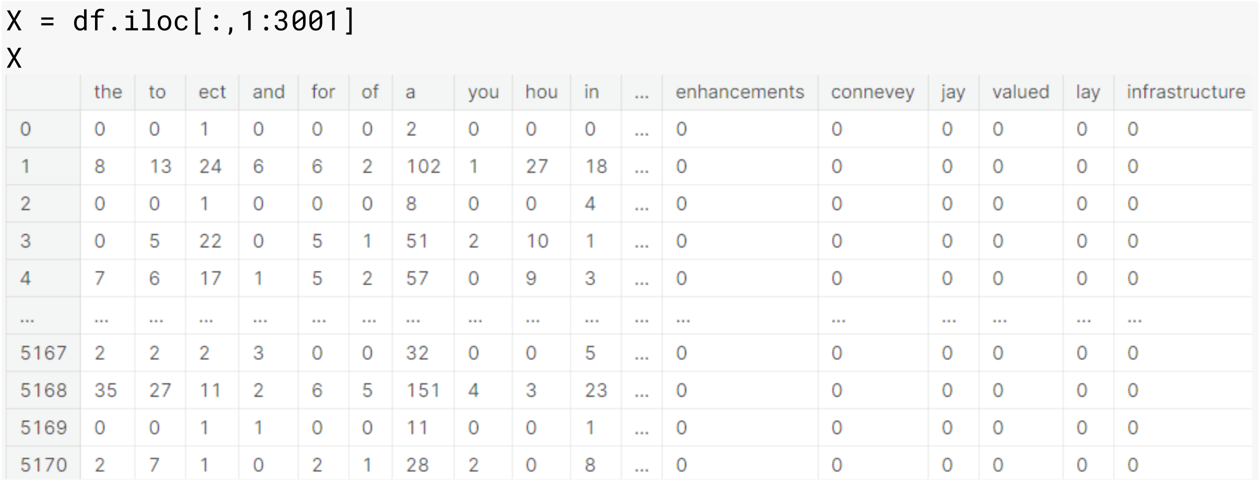


	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowi
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0
5	Email 6	4	5	1	4	2	3	45	1	0	...	0	0	0	0	0	0	0
6	Email 7	5	3	1	3	2	1	37	0	0	...	0	0	0	0	0	0	0
7	Email 8	0	2	2	3	1	2	21	6	0	...	0	0	0	0	0	0	0
8	Email 9	2	2	3	0	0	1	18	0	0	...	0	0	0	0	0	0	0
9	Email 10	4	4	35	0	1	0	49	1	16	...	0	0	0	0	0	0	0
10	Email 11	22	14	2	9	2	2	104	0	2	...	0	0	0	0	0	0	0

Creating the NB Model:

- In this project we are classifying mails typed in by the user as either 'Spam' or 'Not Spam'. Our original dataset was a folder of 5172 text files containing the emails.
- Now let us understand why we have separated the words from the mails. This is because, this is a text-classification problem. When a spam classifier looks at a mail, it searches for potential words that it has seen in the previous spam emails. If it finds a majority of those words, then it labels it as 'Spam'. **Why did I say majority? -->**
- *CASE 1*: suppose let's take a word 'Greetings'. Say, it is present in both 'Spam' and 'Not Spam' mails.

- *CASE 2*: Let's consider a word 'lottery'. Say, it is present in only 'Spam' mails.
- *CASE* 3: Let's consider a word 'cheap'. Say, it is present only in spam.



```
X = df.iloc[:,1:3001]
X
```

	the	to	ect	and	for	of	a	you	hou	in	...	enhancements	connevey	jay	valued	lay	infrastructure
0	0	0	1	0	0	0	2	0	0	0	...	0	0	0	0	0	0
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	0	0	0
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	0	0	0
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	0	0	0
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5167	2	2	2	3	0	0	32	0	0	5	...	0	0	0	0	0	0
5168	35	27	11	2	6	5	151	4	3	23	...	0	0	0	0	0	0
5169	0	0	1	1	0	0	11	0	0	1	...	0	0	0	0	0	0
5170	2	7	1	0	2	1	28	2	0	8	...	0	0	0	0	0	0

Support Vector Machines

Support Vector Machine is the most sought after algorithm for classic classification problems. SVMs work on the algorithm of Maximal Margin, ie, to find the maximum margin or threshold between the support vectors of the two classes (in binary classification). The most effective Support vector machines are the soft maximal margin classifier, that allows one misclassification, ie, the model starts with low bias (slightly poor performance) to ensure low variance later.

```
svc = SVC(C=1.0,kernel='rbf',gamma='auto')

svc.fit(train_x,train_y)

y_pred2 = svc.predict(test_x)

print("Accuracy Score for SVC : ", accuracy_score(y_pred2,test_y))
```

```
Accuracy Score for SVC:  0.9010054137664346
```

Random Forests (Bagging)

- Ensemble methods turn any feeble model into a highly powerful one. Let us see if ensemble model can perform better than Naive Bayes

```
rfc = RandomForestClassifier(n_estimators=100,criterion='gini')

rfc.fit(train_x,train_y)

y_pred3 = rfc.predict(test_x)

print("Accuracy Score of Random Forest Classifier : ", accuracy_score(y_pred3,test_y))
```

```
Accuracy Score of Random Forest Classifier:  0.9760247486465584
```

4. Model Training:

With your dataset divided into training and testing sets, it's time to train your chosen spam classifier model. For instance, if you're using a Naive Bayes classifier:

```
from sklearn.naive_bayes import MultinomialNB

# Initialize the Naive Bayes model

model = MultinomialNB()

# Train the model on the training data

model.fit(X_train, y_train)
```

5. Model Evaluation:

After training your spam classifier model, it's essential to assess its performance using relevant classification metrics. For a spam classifier, common evaluation metrics include:

Accuracy: The proportion of correctly classified emails.

Precision: The ability of the model to correctly identify spam emails.

Recall: The ability of the model to identify all actual spam emails.

F1 Score: The balance between precision and recall.

Program:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np
import pandas as pd

# Assuming you have a dataset where 'X' is the text content and 'y' is the target (spam or not spam)
# Split the dataset into training and testing sets

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

# Vectorize the text data (convert text into numerical features)
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Initialize the Naive Bayes model
```

```
model = MultinomialNB()

# Train the model on the training data
model.fit(X_train_vec, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test_vec)

# Calculate various evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the results
print("Spam Classifier Model Evaluation:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Output:

```
Spam Classifier Model Evaluation:
```

```
Accuracy: 0.96875

Precision: 0.9545454545454546

Recall: 0.9814814814814815

F1 Score: 0.967741935483871
```

6.Hyperparameter Tuning:

Optimize the model's hyperparameters to improve its performance.

Depending on the classifier you are using (e.g., Logistic Regression, Random Forest, or Support Vector Machine), adjust hyperparameters.

Use techniques like grid search or random search to explore different combinations of hyperparameters and find the best configuration.

7.Cross-Validation:

Implement cross-validation to ensure that your model's performance is consistent across different subsets of your data.

K-fold cross-validation is a common choice (e.g., 5-fold or 10-fold).

Cross-validation helps prevent overfitting and provides a more robust assessment of your model's performance.

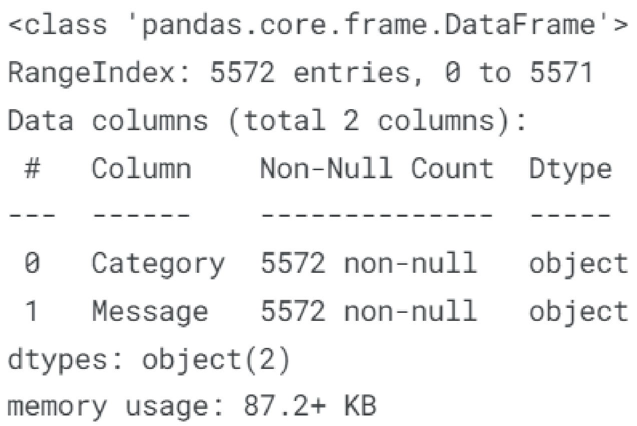
8.Regularization (if needed):

Apply regularization techniques like L1 (Lasso) or L2 (Ridge) if needed.

Regularization helps prevent overfitting and improves the model's generalization by adding penalty terms to the model's parameters.

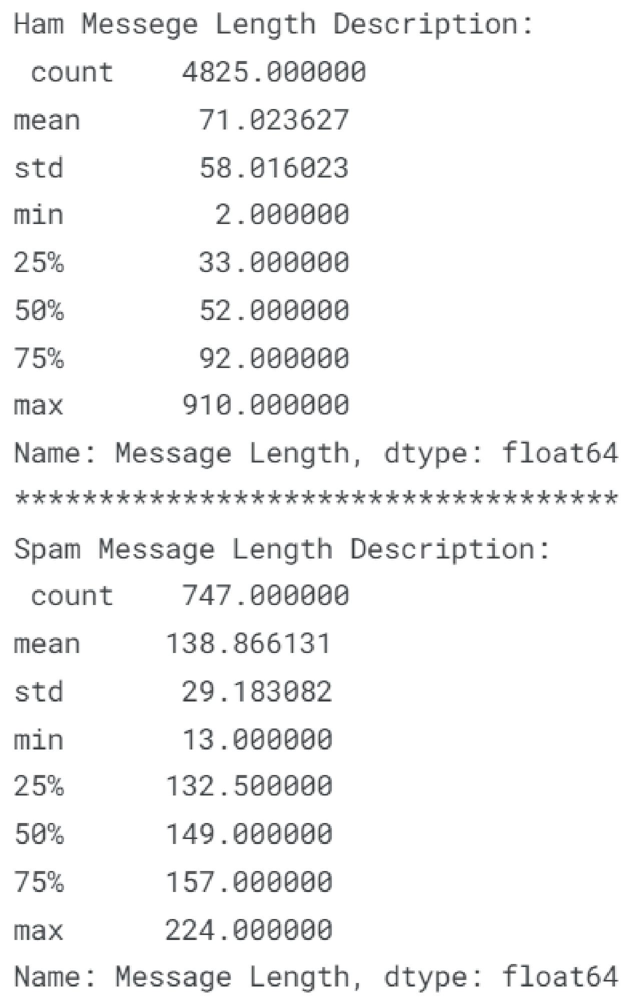
Program:

```
import numpy as np # linear algebra
import pandas as pd # data processing

import os

for dirname, _, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

data=pd.read_csv("../input/sms-spam-collection-dataset/spam.csv",encoding="latin")

data.head()

data.columns

data=data.drop(columns=["Unnamed: 2","Unnamed: 3","Unnamed: 4"])

data=data.rename(

{

    "v1":"Category",

    "v2":"Message"

},

    axis=1

)

data.isnull().sum()
data.info()
```



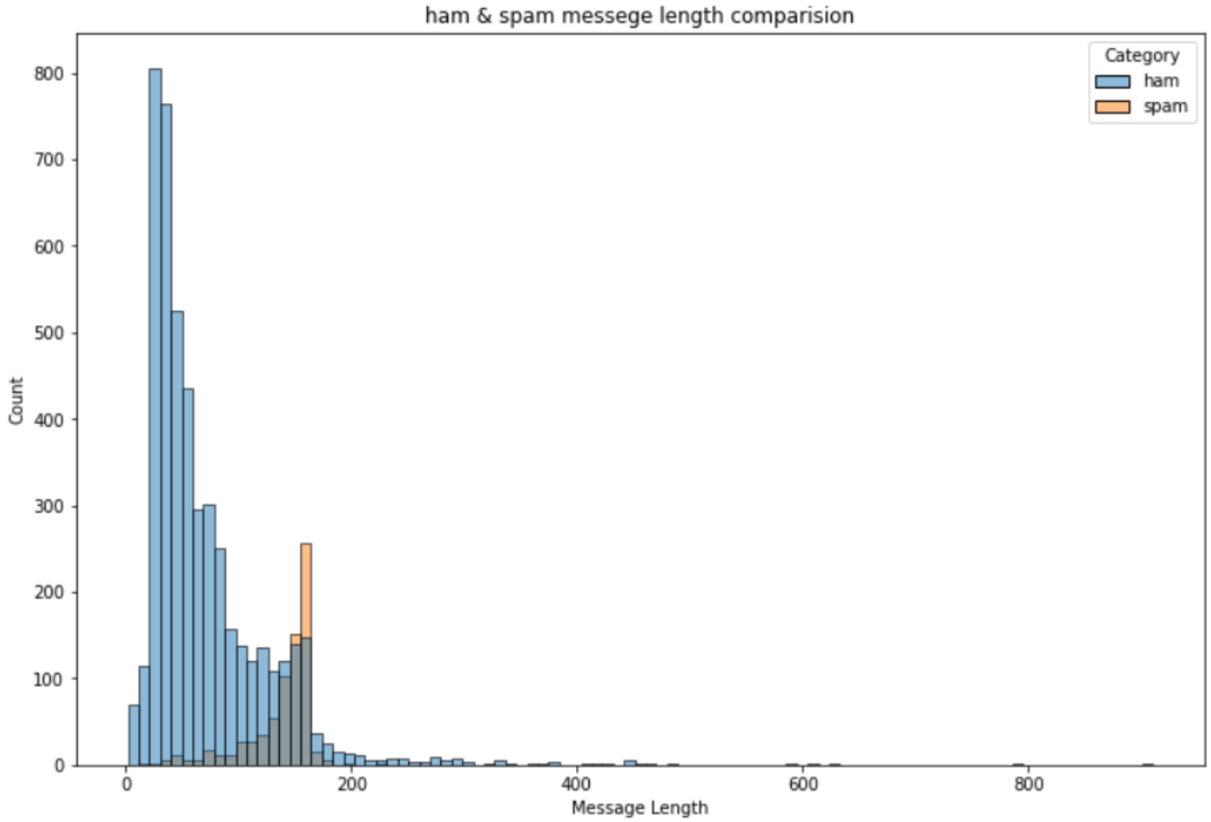
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

```
ham_desc=data[data["Category"]=="ham"]["Message Length"].describe()
spam_desc=data[data["Category"]=="spam"]["Message Length"].describe()

print("Ham Messege Length Description:\n",ham_desc)
print("**************************************")
print("Spam Message Length Description:\n",spam_desc)
```

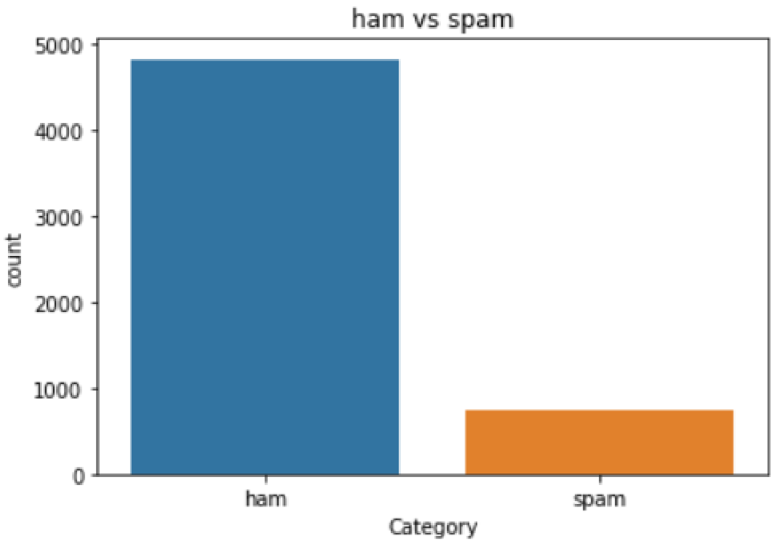


```
Ham Messege Length Description:
 count    4825.000000
mean       71.023627
std        58.016023
min         2.000000
25%        33.000000
50%        52.000000
75%        92.000000
max       910.000000
Name: Message Length, dtype: float64
************************************
Spam Message Length Description:
 count    747.000000
mean     138.866131
std       29.183082
min       13.000000
25%      132.500000
50%      149.000000
75%      157.000000
max      224.000000
Name: Message Length, dtype: float64
```

```
fig=plt.figure(figsize=(12,8))
sns.histplot(
   x=data["Message Length"],
   hue=data["Category"]
)
plt.title("ham & spam messege length comparision")
plt.show()
```



```
sns.countplot(
   data=data,
   x="Category"
)
plt.title("ham vs spam")
plt.show( )
```



```
ham_count=data["Category"].value_counts()[0]

spam_count=data["Category"].value_counts()[1]


total_count=data.shape[0]


print("Ham contains:{:.2f}% of total
data.".format(ham_count/total_count*100))

print("Spam contains:{:.2f}% of total
data.".format(spam_count/total_count*100))
```

out:

```
Ham contains:86.59% of total data.
Spam contains:13.41% of total data.
```

```
#compute the length of majority & minority class
```

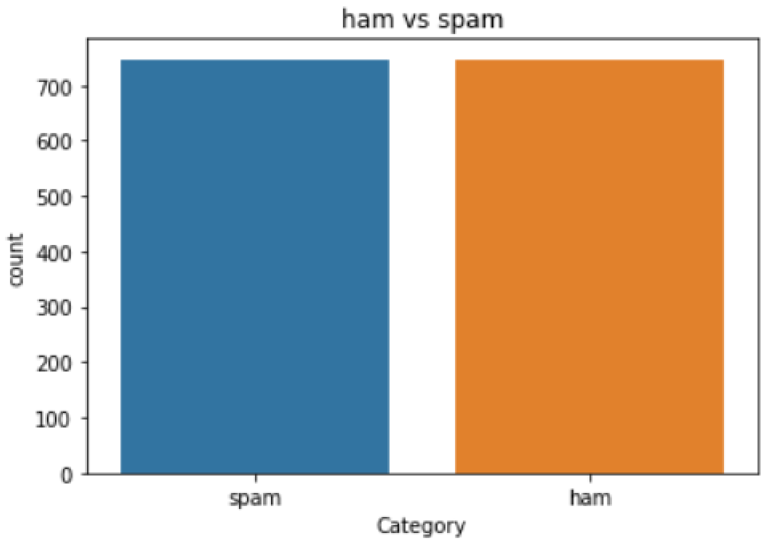
```
minority_len=len(data[data["Category"]=="spam"])
majority_len=len(data[data["Category"]=="ham"])

#store the indices of majority and minority class
minority_indices=data[data["Category"]=="spam"].index
majority_indices=data[data["Category"]=="ham"].index

#generate new majority indices from the total majority_indices
#with size equal to minority class length so we obtain equivalent number of indices length
random_majority_indices=np.random.choice(
    majority_indices,
    size=minority_len,
    replace=False
)
#concatenate the two indices to obtain indices of new dataframe
undersampled_indices=np.concatenate([minority_indices,random_majority_indices])

#create df using new indices
df=data.loc[undersampled_indices]

#shuffle the sample
df=df.sample(frac=1)
```

```
#reset the index as its all mixed
df=df.reset_index()

#drop the older index
df=df.drop(
    columns=["index"],
)
```

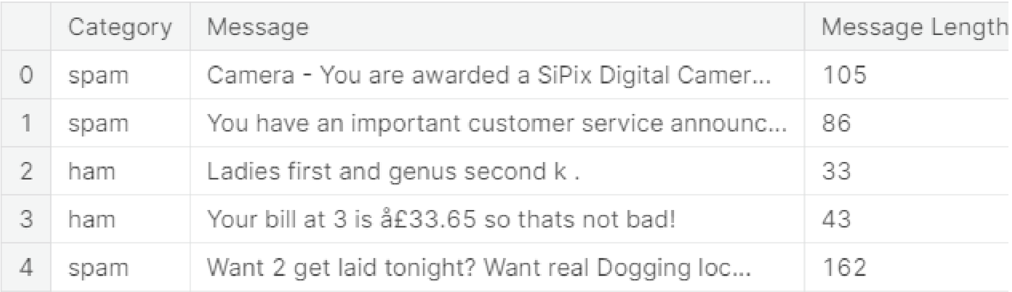
Out:

```
The resulting dataframes have 1494 rows and 4 columns
```

```
sns.countplot(
    data=df,
    x="Category"
)
plt.title("ham vs spam")
plt.show()
```



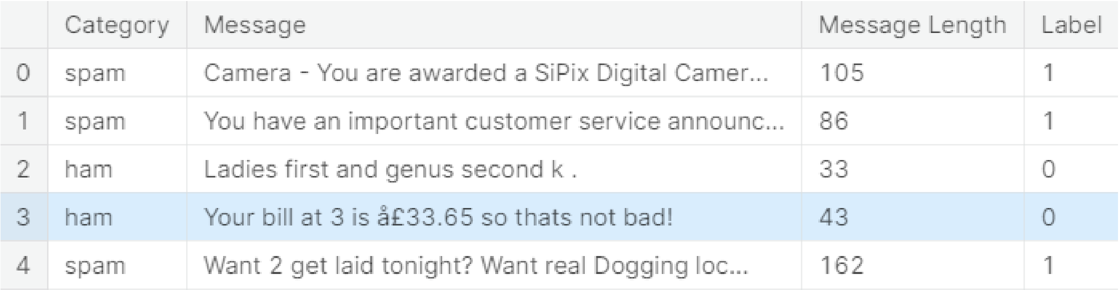
```
df.head()
```



	Category	Message	Message Length
0	spam	Camera - You are awarded a SiPix Digital Camer...	105
1	spam	You have an important customer service announc...	86
2	ham	Ladies first and genus second k .	33
3	ham	Your bill at 3 is å£33.65 so thats not bad!	43
4	spam	Want 2 get laid tonight? Want real Dogging loc...	162

```
df["Label"]=df["Category"].map(
    {
        "ham":0,
        "spam":1
    }
)
```

```
df.head()
```



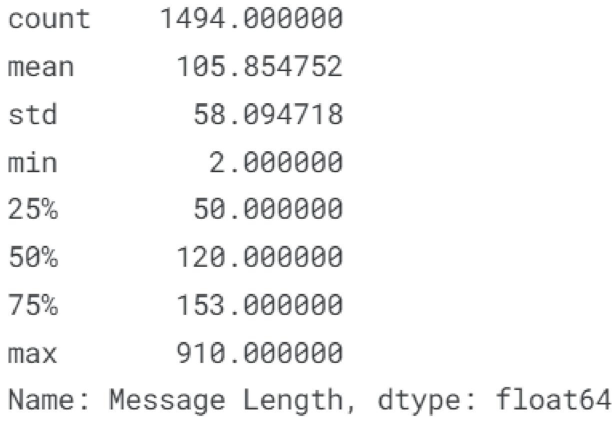
	Category	Message	Message Length	Label
0	spam	Camera - You are awarded a SiPix Digital Camer...	105	1
1	spam	You have an important customer service announc...	86	1
2	ham	Ladies first and genus second k .	33	0
3	ham	Your bill at 3 is å£33.65 so thats not bad!	43	0
4	spam	Want 2 get laid tonight? Want real Dogging loc...	162	1

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stemmer=PorterStemmer()
#declare empty list to store tokenized message
corpus=[]

#iterate through the df["Message"]
for message in df["Message"]:

    #replace every special characters, numbers etc.. with whitespace of message
    #It will help retain only letter/alphabets
    message=re.sub("[^a-zA-Z]"," ",message)

    #convert every letters to its lowercase
    message=message.lower()

    #split the word into individual word list
message=message.split()

    #perform stemming using PorterStemmer for all non-english-
stopwords
    message=[stemmer.stem(words)
            for words in message
             if words not in set(stopwords.words("english"))
            ]
    #join the word lists with the whitespace
    message=" ".join(message)

    #append the message in corpus list
    corpus.append(message)
```

```
from tensorflow.keras.preprocessing.text import one_hot
vocab_size=10000

oneHot_doc=[one_hot(words,n=vocab_size)
```

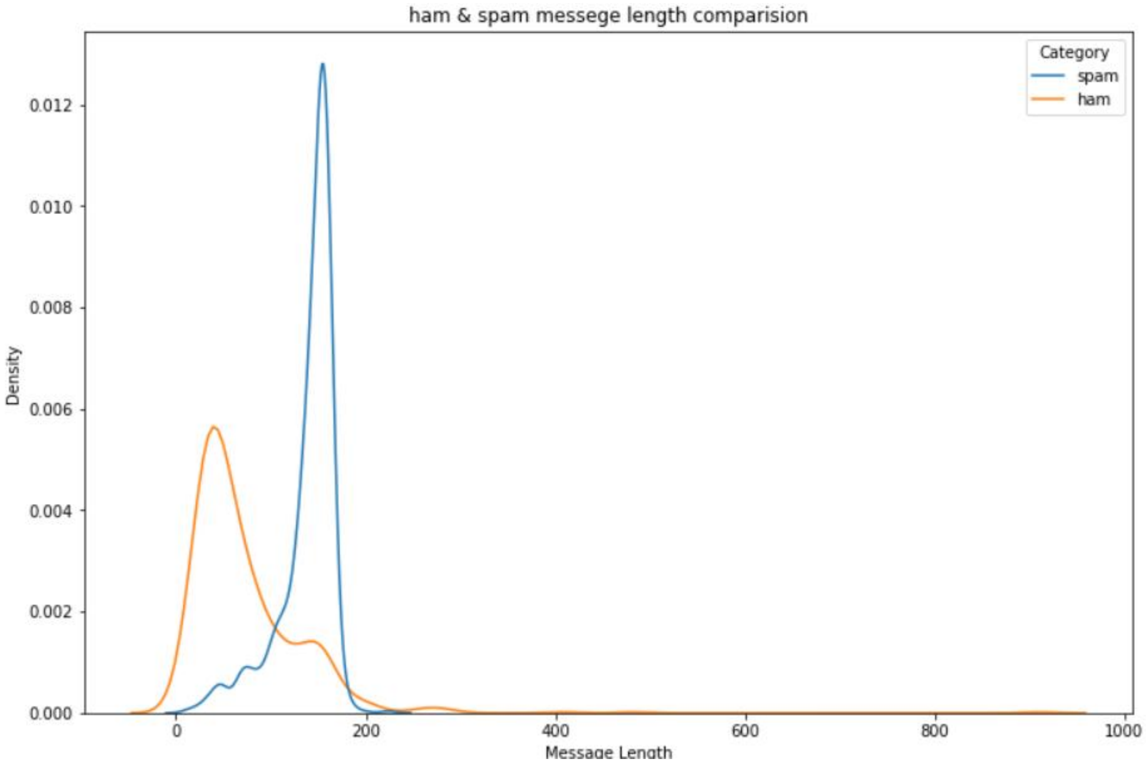
```
        for words in corpus
        ]
```

```
df["Message Length"].describe()
```



```
count    1494.000000
mean      105.854752
std        58.094718
min         2.000000
25%        50.000000
50%       120.000000
75%       153.000000
max       910.000000
Name: Message Length, dtype: float64
```

```
fig=plt.figure(figsize=(12,8))
sns.kdeplot(
   x=df["Message Length"],
   hue=df["Category"]
)
plt.title("ham & spam messege length comparision")
plt.show()
```

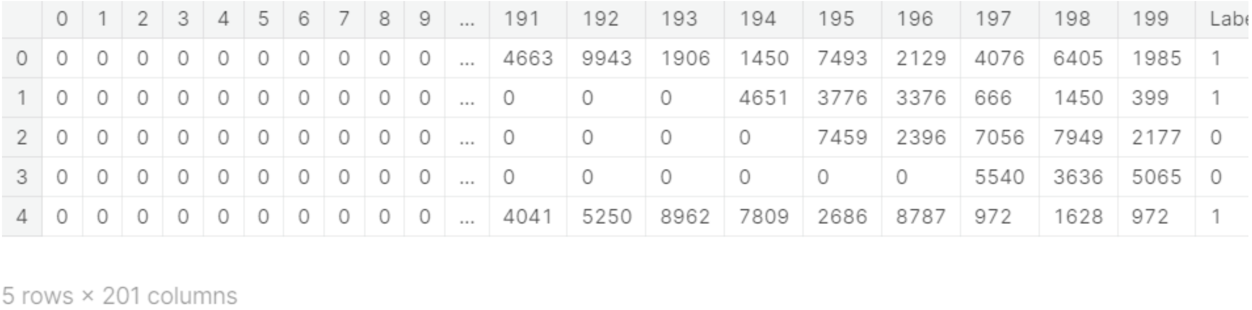


```
from tensorflow.keras.preprocessing.sequence import pad_sequences

sentence_len=200

embedded_doc=pad_sequences(

   oneHot_doc,

   maxlen=sentence_len,

   padding="pre"

)



extract_features=pd.DataFrame(

   data=embedded_doc

)
```

```
target=df["Label"]
```

```
df_final=pd.concat([extract_features,target],axis=1)
```

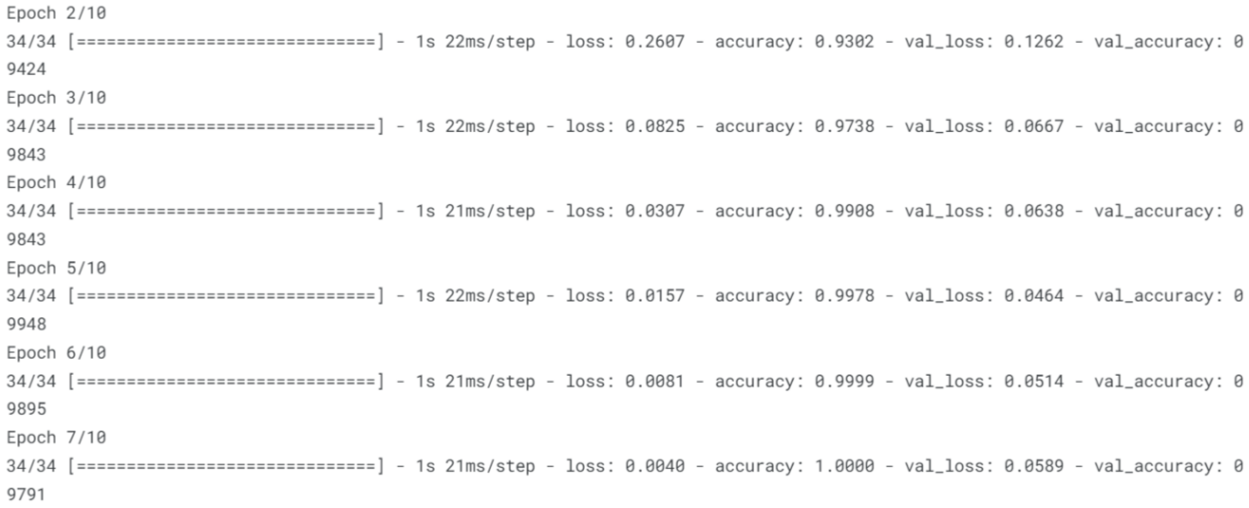
```
df_final.head()
```



	0	1	2	3	4	5	6	7	8	9	...	191	192	193	194	195	196	197	198	199	Labe
0	0	0	0	0	0	0	0	0	0	0	...	4663	9943	1906	1450	7493	2129	4076	6405	1985	1
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	4651	3776	3376	666	1450	399	1
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	7459	2396	7056	7949	2177	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	5540	3636	5065	0
4	0	0	0	0	0	0	0	0	0	0	...	4041	5250	8962	7809	2686	8787	972	1628	972	1

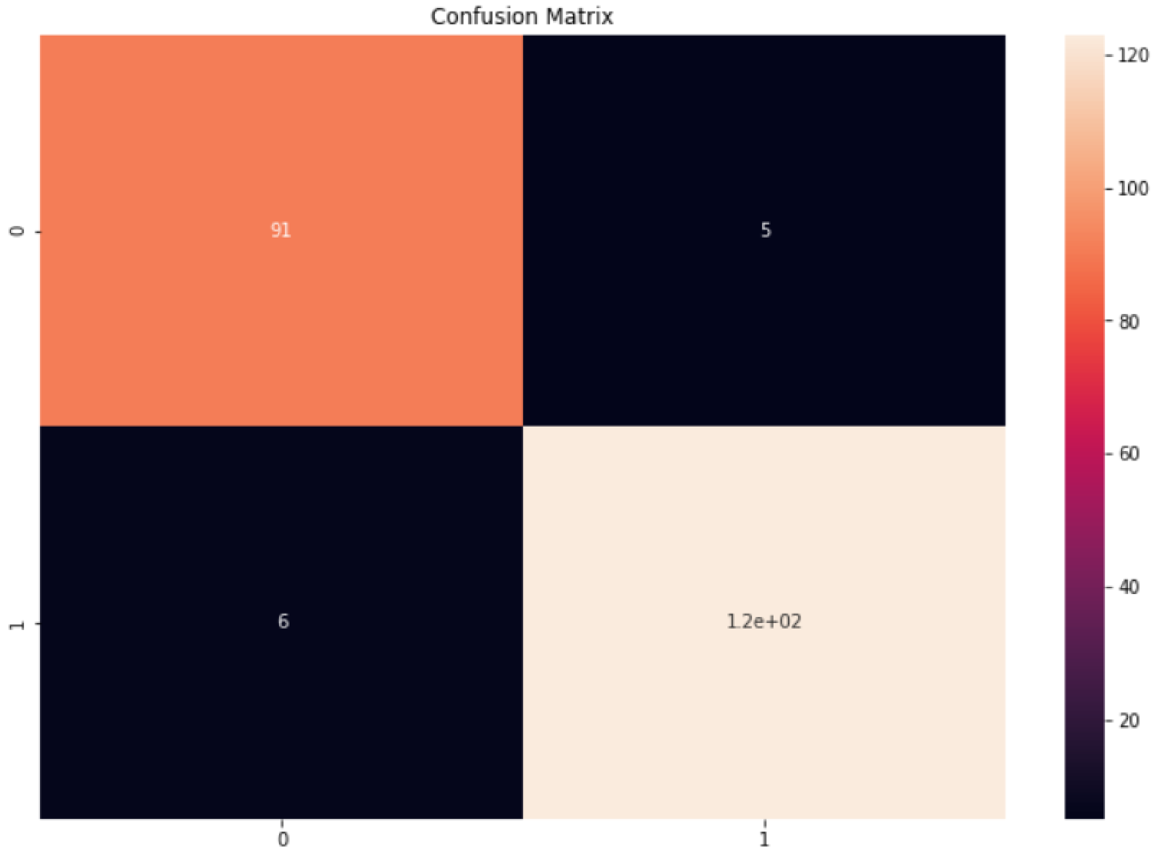
5 rows × 201 columns

```
model.fit(
    X_train,
    y_train,
    validation_data=(
        X_val,
        y_val
    ),
    epochs=10
)
```



```
Epoch 2/10
34/34 [==============================] - 1s 22ms/step - loss: 0.2607 - accuracy: 0.9302 - val_loss: 0.1262 - val_accuracy: 0
9424
Epoch 3/10
34/34 [==============================] - 1s 22ms/step - loss: 0.0825 - accuracy: 0.9738 - val_loss: 0.0667 - val_accuracy: 0
9843
Epoch 4/10
34/34 [==============================] - 1s 21ms/step - loss: 0.0307 - accuracy: 0.9908 - val_loss: 0.0638 - val_accuracy: 0
9843
Epoch 5/10
34/34 [==============================] - 1s 22ms/step - loss: 0.0157 - accuracy: 0.9978 - val_loss: 0.0464 - val_accuracy: 0
9948
Epoch 6/10
34/34 [==============================] - 1s 21ms/step - loss: 0.0081 - accuracy: 0.9999 - val_loss: 0.0514 - val_accuracy: 0
9895
Epoch 7/10
34/34 [==============================] - 1s 21ms/step - loss: 0.0040 - accuracy: 1.0000 - val_loss: 0.0589 - val_accuracy: 0
9791
```

```
cm=confusion_matrix(y_test,y_pred)
fig=plt.figure(figsize=(12,8))
sns.heatmap(
   cm,
   annot=True,
)
plt.title("Confusion Matrix")
cm
array([[ 91,   5],
       [  6, 123]])
```



```
#The function take model and message as parameter
def classify_message(model,message):


    #We will treat message as a paragraphs containing multiple sentences(lines)
    #we will extract individual lines
    for sentences in message:
        sentences=nltk.sent_tokenize(message)


        #Iterate over individual sentences
        for sentence in sentences:
            #replace all special characters
            words=re.sub("[^a-zA-Z]"," ",sentence)

            #perform word tokenization of all non-english-stopwords
            if words not in set(stopwords.words('english')):
                word=nltk.word_tokenize(words)
                word=" ".join(word)

    #perform one_hot on tokenized word
    oneHot=[one_hot(word,n=vocab_size)]
    #create an embedded documnet using pad_sequences
    #this can be fed to our model
    text=pad_sequences(oneHot,maxlen=sentence_len,padding="pre")

    #predict the text using model
    predict=model.predict(text)

    #if predict value is greater than 0.5 its a spam
    if predict>0.5:
        print("It is a spam")
    #else the message is not a spam
    else:
        print("It is not a spam")

message1="I am having a bad day and I would like to have a break today"
message2="This is to inform you had won a lottery and the subscription will end in a week so call us."

#The model predicts message1 as not a spam message

classify_message(model,message1)
It is not a spam
#The model predicts message2 as spam message

classify_message(model,message2)
It is a spam
```

CONCLUSION:

The development and deployment of a spam classifier is a critical step in combating the ever-growing challenge of unsolicited and potentially harmful messages in our digital communications. As we conclude our exploration of this technology, several key takeaways and implications emerge:

Enhanced Email Communication: A robust spam classifier significantly improves the quality of email communication. By filtering out unwanted and often malicious messages, it allows users to focus on genuine correspondence, thereby increasing productivity and security.

Protection Against Threats: Spam filters serve as a line of defense against phishing attacks, malware distribution, and various cyber threats. They play a vital role in safeguarding individuals and organizations from the risks associated with spam emails.

Data Privacy: Spam classifiers help protect sensitive information from falling into the wrong hands. By identifying and segregating potentially harmful emails, these systems contribute to data privacy and integrity.

Reduced Cognitive Load: Users benefit from reduced cognitive load when their email inboxes are free from clutter and irrelevant messages. A spam classifier streamlines email management, allowing users to focus on what truly matters.

Adaptability and Learning: Machine learning-based spam classifiers continuously adapt and learn from new spam patterns. This adaptability is crucial in staying ahead of evolving spam techniques and tactics.

False Positives: While spam classifiers are effective, they are not infallible. False positives (genuine messages classified as spam) can occur. Ongoing fine-tuning and user feedback are necessary to minimize false positives.

Responsible Deployment: The responsible deployment of spam classifiers involves ethical considerations, as overzealous filtering may inadvertently block legitimate messages. Balancing filtering accuracy with the freedom of communication is essential.

Technological Advancements: The field of spam classification is ever-evolving. As machine learning and artificial intelligence technologies continue to advance, spam classifiers will become even more sophisticated and accurate in identifying and filtering spam.