Spam Email Classifier Using NLP

A PROJECT REPORT

Submitted by

Abdullah Khan (21BCS10510)

Loga Aswin (21BCS10573)

Sahil Sharma (21BCS6732)

in partial fulfilment for the award of the degree of

BACHELOR'S OF ENGINEERING

IN

CSE – SPECIALIZATION IN AI / ML

Chandigarh University

April 2024

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report “Prediction Of Cyber Security Attacks” is the bonafide work of **“Loga Aswin, Abdullah Khan, Sahil Sharma”** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Er. Aman Kaushik	Mr. Jaswinder Singh
HEAD OF THE DEPARTMENT	**SUPERVISOR**
AIT - CSE	ASSISTANT PROFESSOR
	AIT- CSE

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who have supported and contributed to the completion of this project report on prediction of cyber security attacks. Your invaluable assistance and encouragement have been instrumental in its success.

We extend our heartfelt appreciation to our supervisor, Mr. Jaswinder Singh, for his guidance, expertise, and valuable feedback throughout the project. His support has been invaluable in shaping the direction of our research and enhancing its quality.

We would like to thank the faculty members of Chandigarh University for providing us with a conducive academic environment and access to resources. Their knowledge and expertise in the field of e-resource technology have greatly enriched our understanding.

We are grateful to the participants of this study for their cooperation and willingness to share their experiences. Their insights have significantly contributed to the findings and conclusions presented in this report.

We would also like to acknowledge the authors of the referenced literature, whose work has served as a foundation for our research. Their contributions have provided valuable insights and frameworks.

Our appreciation extends to our families and friends for their unwavering support and understanding throughout this project. Their encouragement has been a constant source of motivation.

Lastly, we would like to express our gratitude to all those who have provided direct or indirect assistance during this project. Your contributions have made a significant impact, and we are sincerely thankful for your support.

In conclusion, we would like to acknowledge the above-mentioned individuals and groups for their invaluable contributions to the completion of this project report. Your support has been instrumental, and we are genuinely grateful for your presence in our academic journey.

Thank you.

TABLE OF CONTENTS

ABSTRACT

Email is still the major method of communication and cooperation in a world where digital communication and connectivity rule. But this reliance on email also means that there's always a risk of spam, which clogs inboxes and jeopardizes privacy and data security. In order to efficiently forecast and combat spam emails, the "Email Spam Classifier" project uses cutting-edge machine Learning-algorithms.

The project's core is the merging of data analysis with artificial intelligence, which gives the system the ability to identify subtle trends and telltale signs of spam in email content. The project builds strong prediction models that can recognize and filter out spam by carefully gathering data from a variety of sources, such as email logs, sender reputation databases, and content analysis.

The project's scope includes both real-time monitoring and historical data analysis, enabling ongoing learning and response to changing spam techniques. The project aims to give people and businesses the tools they need to protect their email communication channels by giving users proactive spam detection capabilities.

The "Email Spam Classifier" project is a valuable resource for fostering a more secure and safe digital ecosystem in a setting where email spam is a constant danger. The project's objectives are to preserve sensitive information, increase user productivity, and promote confidence in email communication by anticipatorily recognizing and eliminating spam. The initiative aims to change the email security environment by using proactive machine learning, providing a look into a future when spam is no longer a problem.

List of Figures

List of Tables

Abbreviations

1. **ESC - Email Spam Classifier**
2. **AIS - Artificial Intelligence for Spam**
3. **MLD - Machine Learning for Detection**
4. **DAA - Data Analysis for Antispam**
5. **LCA - Logistic Classification Algorithm**
6. **BAE - Bayesian Email Filtering**
7. **NLP - Natural Language Processing for Spam Detection**
8. **RFS - Random Forest Spam Classifier**
9. **DFM - Deep Feature Mining**
10. **TIR - Textual Information Retrieval**
11. **SMD - Spam Message Detection**
12. **SFC - Spam Filtering Capability**
13. **SCA - Spam Content Analysis**
14. **TLD - Threshold Learning for Detection**
15. **ESD - Email Security Decision-making**

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Email communication has become an integral part of our daily lives, serving as a primary means of professional and personal correspondence. However, along with the convenience it offers, email communication also faces the challenge of spam. Spam emails, often unsolicited and irrelevant, clutter our inboxes, consuming valuable time and resources. To combat this issue, the development of efficient email spam classifiers using Machine Learning (ML) models with Natural Language Processing (NLP) techniques has gained significant attention.

1.1.1 The Machine Learning System

The heart of our approach lies in the utilization of machine learning systems to distinguish between legitimate emails and spam. Machine learning systems, particularly supervised learning algorithms, have shown remarkable effectiveness in solving classification problems by learning patterns and features from labelled data.

In the context of email spam classification, the machine learning system analyzes various attributes and characteristics of emails to make predictions about their nature. These attributes may include the frequency of certain words or phrases, the presence of specific patterns or structures, and metadata such as sender information and timestamps.

By leveraging the power of machine learning, our system can adapt and improve its classification accuracy over time as it encounters new data and refines its understanding of what constitutes spam.

1.1.2 Fundamental

Before delving into the intricacies of email spam classification, it is essential to understand the fundamental concepts underpinning our approach.

At its core, our system relies on Natural Language Processing (NLP) techniques to process and analyze the textual content of emails. NLP encompasses a wide range of methods and algorithms designed to understand and interpret human language, enabling computers to extract meaning and insights from text data.

Within the realm of email spam classification, NLP techniques enable our system to:

Tokenize and preprocess text: Breaking down the textual content of emails into individual words or tokens and performing tasks such as removing stopwords, stemming, and lemmatization to standardize the text for analysis.

Extract features: Identifying relevant features and attributes from the preprocessed text, such as the frequency of specific words or the presence of certain patterns, which serve as inputs to our machine learning model.

Model text representations: Representing the textual content of emails in a format suitable for machine learning algorithms, such as bag-of-words representations or more sophisticated embeddings derived from techniques like Word2Vec or BERT.

By integrating NLP techniques into our machine learning system, we can effectively analyze and classify emails based on their textual content, thereby mitigating the impact of spam on email users and enhancing overall communication efficiency.

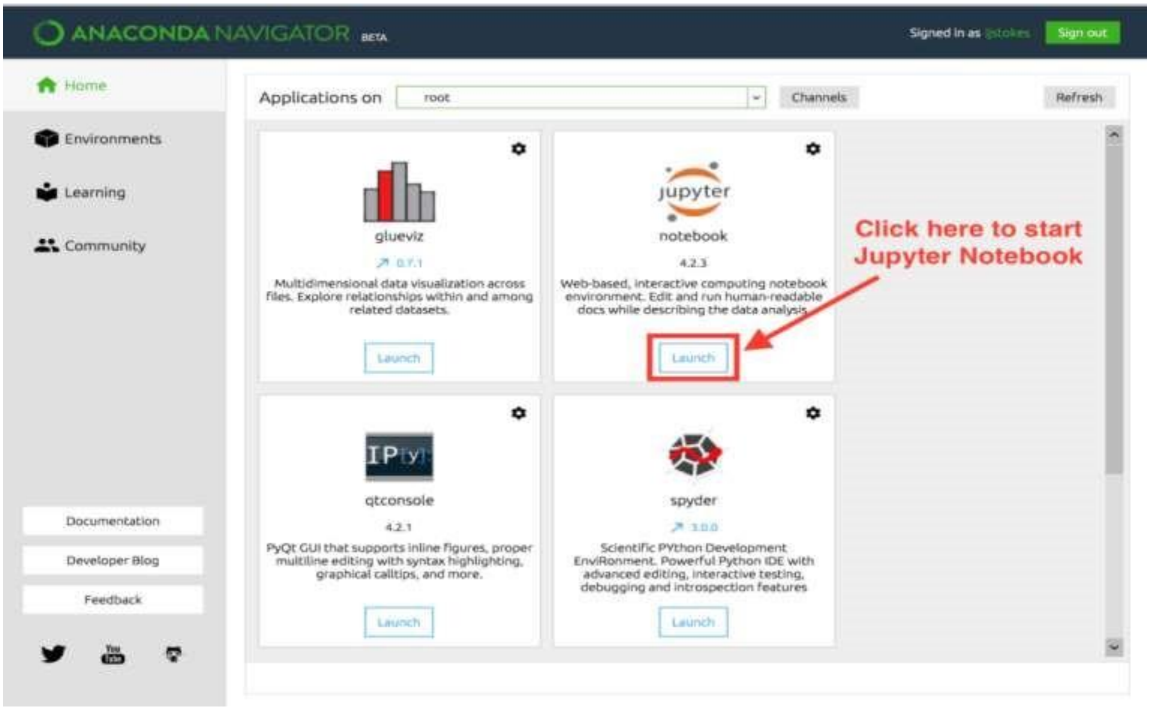
1.2 Jupyter

Jupyter is a popular open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It supports various programming languages, including Python, R, and Julia, making it an excellent tool for data science and machine learning projects.

For your project, Jupyter can serve as a versatile environment for developing, prototyping, and documenting your email spam classifier using machine learning and NLP techniques. Here's how you can leverage Jupyter for different aspects of your project:

Code Development: Jupyter provides an interactive interface where you can write and execute code in cells. You can use Python libraries such as scikit-learn, NLTK (Natural Language Toolkit), and TensorFlow to implement machine learning algorithms and NLP techniques for email spam classification. The ability to run code interactively allows you to experiment with different algorithms, parameters, and data preprocessing steps in real-time.

Data Exploration and Visualization: Jupyter notebooks support inline plotting, allowing you to visualize data distributions, feature correlations, and model performance metrics using libraries like Matplotlib and Seaborn. Visualizations help you gain insights into your dataset and evaluate the effectiveness of your machine learning models.



Multidimensional data visualization across files. Explore relationships within and among related datasets.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

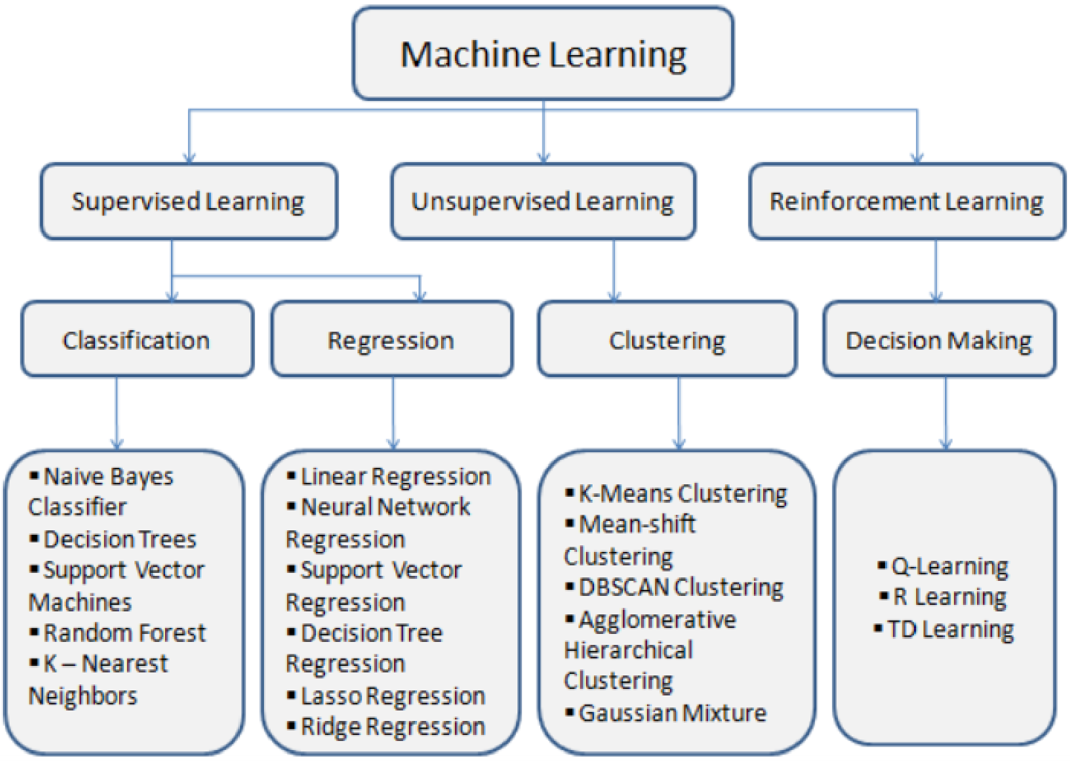
Click here to start Jupyter Notebook

Documentation and Communication: Jupyter notebooks combine code, visualizations, and narrative text in a single document, making it easy to document your project's progress, methodology, and results. You can write explanations, insights, and observations alongside your code cells, providing context for readers and collaborators. Additionally, Jupyter notebooks can be exported to various formats, including HTML, PDF, and slides, enabling you to share your findings and present your project to others.

Overall, Jupyter serves as a powerful tool for data exploration, model development, and project documentation, facilitating the entire workflow of your email spam classifier project.

1.3 Machine Learning

Machine learning is a subfield of artificial intelligence that focuses on developing algorithms and techniques that enable computers to learn from data and make predictions or decisions without being explicitly programmed. In the context of your project on email spam classification, machine learning plays a crucial role in building a predictive model that can automatically classify emails as either spam or legitimate.



Here's a detailed explanation of how machine learning relates to your project:

Supervised Learning: Email spam classification is typically approached as a supervised learning problem, where the algorithm learns to classify emails based on labeled training data. In supervised learning, the algorithm is trained on a dataset containing examples of both spam and legitimate emails, along with their corresponding labels (i.e., whether each email is spam or not). The algorithm then learns patterns and features from the labeled data to make predictions on new, unseen emails.

Feature Extraction: Machine learning models for email spam classification rely on extracting relevant features from the textual content of emails. These features may include word frequencies, presence of specific keywords or phrases, textual patterns, and metadata such as sender information and email structure. Feature extraction techniques preprocess the raw text data and transform it into a numerical representation that can be fed into the machine learning algorithm.

Algorithm Selection: There are various machine learning algorithms that can be used for email spam classification, including but not limited to:

Naive Bayes: A probabilistic classifier based on Bayes' theorem that is commonly used for text classification tasks like spam filtering.

Support Vector Machines (SVM): A supervised learning algorithm that separates data points into different classes using hyperplanes in a high-dimensional space.

Decision Trees and Random Forests: Ensemble learning methods that build a collection of decision trees to improve classification accuracy.

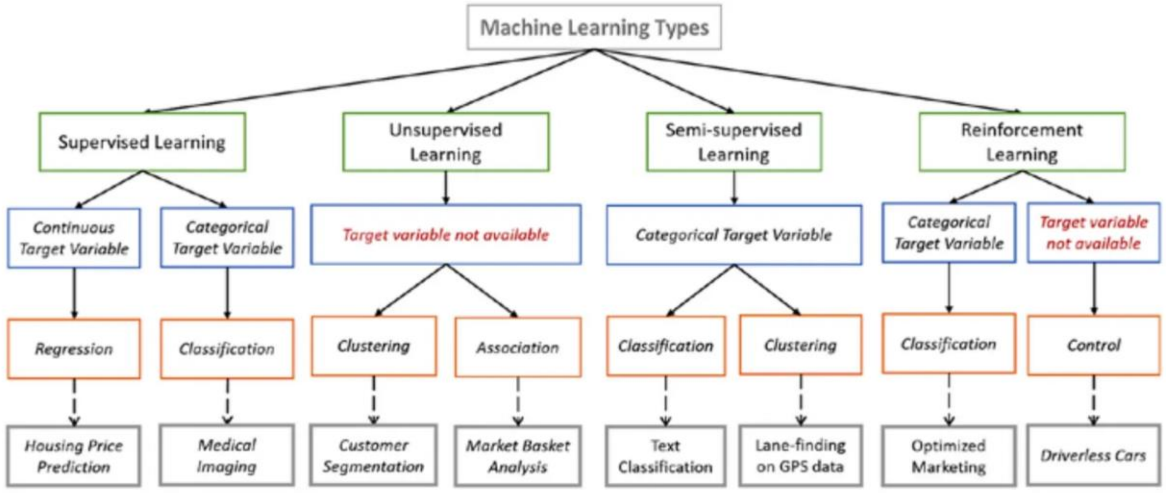
Neural Networks: Deep learning models that consist of interconnected layers of neurons and are capable of learning complex patterns from data.

Model Evaluation and Validation: Once the machine learning model is trained, it needs to be evaluated and validated to assess its performance and generalization ability. Performance metrics such as accuracy, precision, recall, and F1-score are commonly used to evaluate the effectiveness of the classifier. Additionally, techniques like cross-validation and hyperparameter tuning are employed to optimize the model's performance and ensure robustness.

Deployment and Integration: After building and validating the machine learning model, it can be deployed into production environments where it can classify incoming emails in real-time. Integration with email servers or applications allows the classifier to automatically filter out spam emails before they reach users' inboxes, thereby improving email security and user experience.

In summary, machine learning forms the foundation of your email spam classifier project, enabling you to develop a predictive model that can effectively distinguish between spam and legitimate emails based on their textual content.

By leveraging machine learning algorithms and techniques, you can build a scalable and efficient solution to combat the problem of email spam.



1.4 Natural Language Processing

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful. In the context of your project on email spam classification using machine learning with NLP techniques, NLP plays a crucial role in preprocessing and analyzing the textual content of emails to extract relevant features and patterns for classification. Let's delve into the various aspects of NLP that are relevant to your project:

1. Text Preprocessing:

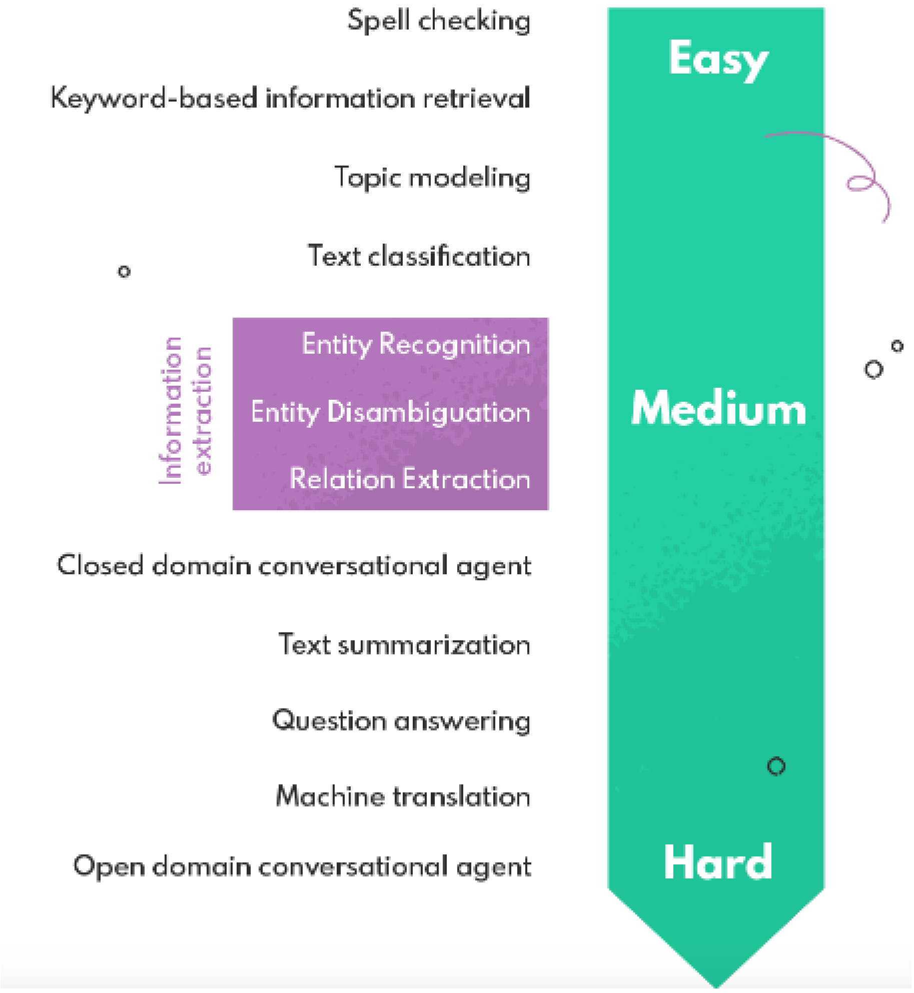
Before applying machine learning algorithms to the textual content of emails, it's essential to preprocess the text to standardize it and make it suitable for analysis. Text preprocessing tasks commonly employed in NLP include:

Tokenization: Breaking down the text into individual words or tokens.

Stopword Removal: Filtering out common words (e.g., "the," "is," "and") that do not carry significant meaning.

Stemming and Lemmatization: Reducing words to their root form to normalize variations (e.g., "running" to "run").

Removing Punctuation and Special Characters: Eliminating non-alphanumeric characters that do not contribute to the meaning of the text.



2. Feature Extraction:

NLP techniques are used to extract relevant features from the preprocessed text data, which serve as inputs to machine learning algorithms for email spam classification. Common feature extraction methods in NLP include:

Bag-of-Words (BoW): Representing the textual content of emails as a sparse matrix of word counts, where each row corresponds to an email and each column corresponds to a unique word in the vocabulary.

TF-IDF (Term Frequency-Inverse Document Frequency): Weighing the importance of words in emails based on their frequency within the email and across the entire dataset, with higher weights assigned to rare words that are more discriminative.

Word Embeddings: Representing words as dense, low-dimensional vectors in a continuous vector space, capturing semantic relationships between words. Techniques such as Word2Vec, GloVe, and FastText generate word embeddings that encode semantic information about words based on their context in large text corpora.

3. Sentiment Analysis:

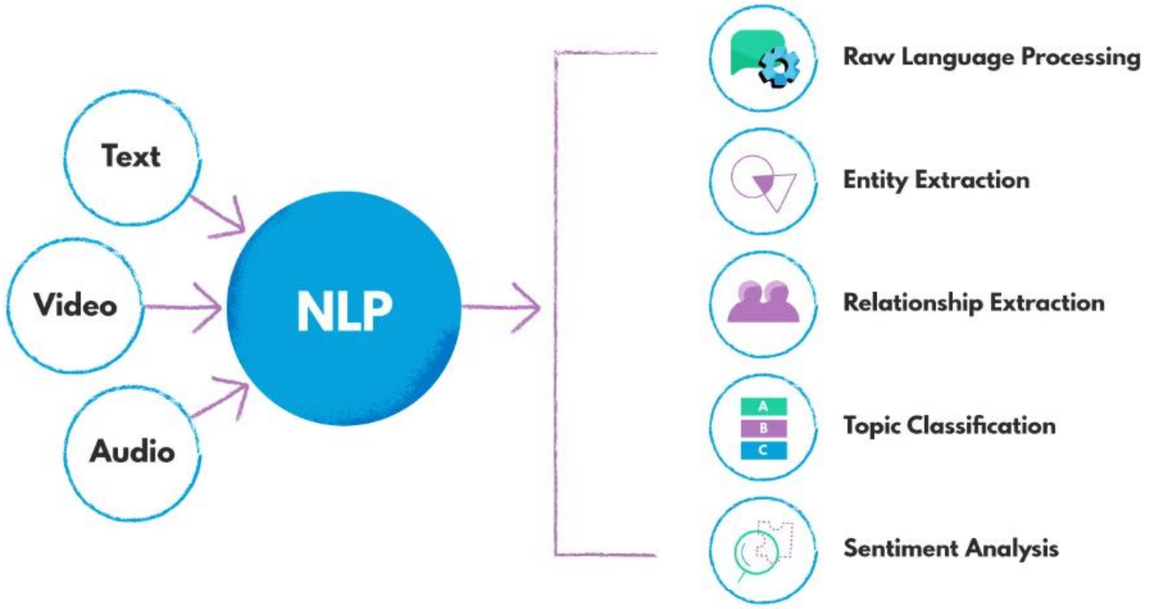
NLP techniques can be used to analyze the sentiment or tone of emails, which may provide additional insights for email spam classification. Sentiment analysis algorithms classify text as positive, negative, or neutral based on the emotional polarity expressed in the text. While sentiment analysis may not be the primary focus of your project, it could be leveraged as an additional feature to enhance the classifier's performance.

4. Named Entity Recognition (NER):

NER is a task in NLP that involves identifying and classifying named entities (e.g., persons, organizations, locations) mentioned in the text. While not directly related to email spam classification, NER could be utilized to extract metadata from emails, such as sender names, email addresses, and company names, which may provide valuable contextual information for classification.

5. Language Models:

Language models, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformer-based architectures like BERT (Bidirectional Encoder Representations from Transformers), are powerful NLP models that can learn complex patterns and relationships in text data. Pretrained language models, in particular, have been shown to achieve state-of-the-art performance on various NLP tasks, including text classification and sentiment analysis. Fine-tuning pretrained language models on email data may enhance the performance of your email spam classifier by leveraging the model's understanding of natural language semantics and syntax.



In summary, Natural Language Processing (NLP) provides a rich set of techniques and methodologies for processing and analyzing textual data, which are essential for building an effective email spam classifier. By leveraging NLP techniques for text preprocessing, feature extraction, and semantic analysis, you can develop a robust machine learning model capable of accurately classifying emails as spam or legitimate based on their textual content.

1.5 CLASSIFICATION TECHNIQUES

Classification techniques play a pivotal role in the development of the "Email Spam Classifier using ML Model with NLP Techniques" project. This section explores various classification methods, focusing on their applicability and effectiveness in distinguishing between spam and legitimate emails.

1.5.1 Neural Network and Deep Learning

Neural networks, particularly deep learning models, have gained significant attention and popularity in recent years due to their remarkable capability to learn complex patterns and representations from data. In the context of email spam classification, neural networks offer a powerful framework for building sophisticated classifiers that can effectively capture the nuanced characteristics of spam emails.

Neural Network Architectures: Various neural network architectures can be employed for email spam classification, including:

Feedforward Neural Networks (FNNs): Basic neural networks consisting of input, hidden, and output layers, suitable for simple classification tasks.

Convolutional Neural Networks (CNNs): CNNs excel at capturing spatial dependencies in data, making them well-suited for tasks involving image classification and sequential data processing, such as email text.

Recurrent Neural Networks (RNNs): RNNs are designed to handle sequential data by maintaining a memory state across time steps, making them suitable for processing email text with temporal dependencies.

Long Short-Term Memory (LSTM) Networks and Gated Recurrent Units (GRUs): Specialized RNN architectures equipped with mechanisms to mitigate the vanishing gradient problem and capture long-range dependencies in sequences.

Deep Learning Techniques: In addition to traditional neural network architectures, deep learning techniques offer several advantages for email spam classification, including:

Word Embeddings: Pretrained word embeddings, such as Word2Vec or GloVe, can be used to represent words as dense vectors, capturing semantic relationships between words in email text.

Attention Mechanisms: Attention mechanisms allow neural networks to focus on relevant parts of the input sequence, enabling them to effectively process long documents such as email bodies.

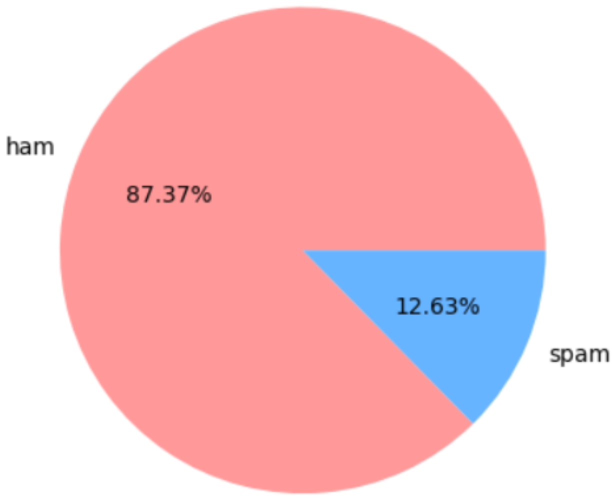
Transfer Learning: Transfer learning techniques, such as fine-tuning pretrained language models like BERT or GPT, leverage knowledge learned from large text corpora to improve the performance of email spam classifiers with limited labeled data.

1.5.2 Methodologies – Given Input and Expected Output

In the context of email spam classification, the methodologies employed typically involve preprocessing textual data, extracting relevant features, and mapping emails to their corresponding class labels (spam or legitimate). The following methodologies are commonly used:

Preprocessing Techniques: Textual data preprocessing is essential for preparing email content for classification. Common preprocessing techniques include:

[Fig. 3.1 Spam and Ham Email Percentage]

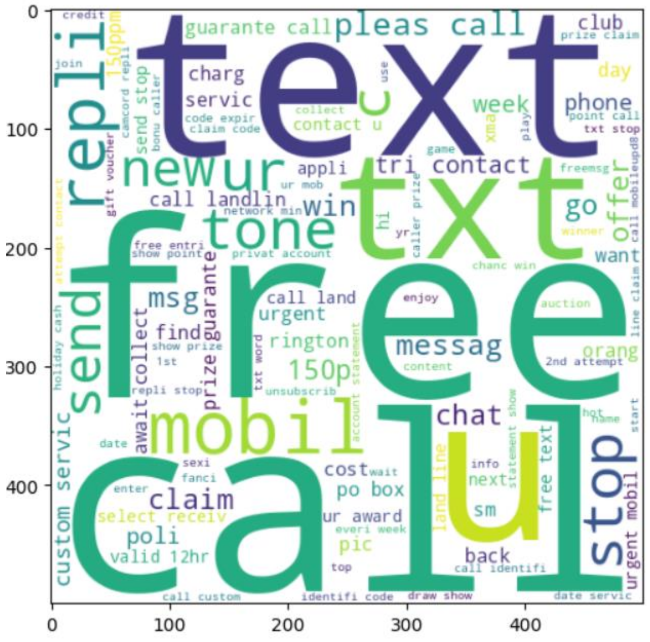


Tokenization: Splitting the text into individual words or tokens.

Stopword Removal: Filtering out common words that do not carry significant meaning (e.g., "the," "is," "and").

Stemming and Lemmatization: Reducing words to their base or root forms to normalize variations (e.g., "running" to "run").

Feature Extraction: Feature extraction techniques aim to capture relevant information from the preprocessed text data, enabling the classification model to make accurate predictions. Common feature extraction methods include:

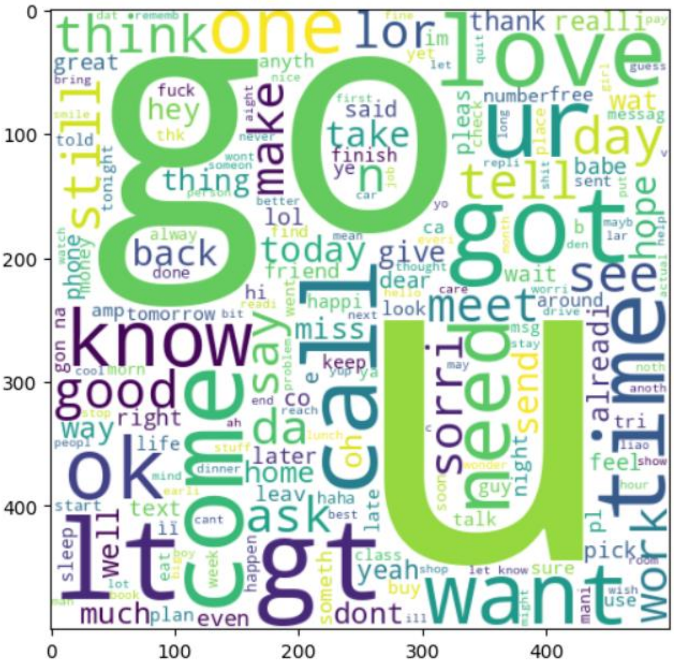


Bag-of-Words (BoW): Representing the email content as a vector of word counts or frequencies, disregarding word order and context.

Term Frequency-Inverse Document Frequency (TF-IDF): Weighing the importance of words in emails based on their frequency within the email and across the entire dataset.

Word Embeddings: Representing words as dense, low-dimensional vectors in a continuous vector space, capturing semantic relationships between words.

Classification Algorithms: Once the textual data is preprocessed and features are extracted, various classification algorithms can be applied to map emails to their corresponding class labels. Commonly used classification algorithms for email spam classification include:



Naive Bayes: A probabilistic classifier based on Bayes' theorem, known for its simplicity and efficiency in text classification tasks.

Support Vector Machines (SVM): A supervised learning algorithm that separates data points into different classes using hyperplanes in a high-dimensional space.

Decision Trees and Random Forests: Ensemble learning methods that build a collection of decision trees to improve classification accuracy.

Neural Networks: Deep learning models capable of learning complex patterns and representations from data, particularly effective for tasks involving sequential data processing like email text classification.

Evaluation Metrics: To assess the performance of the email spam classifier, various evaluation metrics are employed to measure its accuracy, precision, recall, F1-score, and area under the ROC curve. These metrics provide quantitative measures of the classifier's effectiveness in distinguishing between spam and legitimate emails.

1.6 Objective and Scope of Project

Objective:

The primary objective of the "Email Spam Classifier using ML Model with NLP Techniques" project is to develop a robust and accurate classifier capable of distinguishing between spam and legitimate emails. The project aims to leverage machine learning (ML) models integrated with natural language processing (NLP) techniques to enhance email security and improve user experience by automatically filtering out unwanted spam emails.

Scope:

The scope of the project encompasses the following key aspects:

Development of ML-based Classifier: The project involves designing, implementing, and evaluating a machine learning classifier that can effectively identify and classify emails as either spam or legitimate. The classifier will be trained on a labeled dataset of emails, utilizing NLP techniques to extract relevant features from the textual content of emails.

Integration of NLP Techniques: Natural language processing techniques will be employed to preprocess the textual data, tokenize, remove stopwords, and extract meaningful features that capture the distinguishing characteristics of spam emails. NLP methods such as bag-of-words, TF-IDF, and word embeddings will be explored to represent the textual content of emails in a format suitable for ML model training.

Evaluation and Validation: The project includes thorough evaluation and validation of the developed classifier to assess its performance, accuracy, precision, recall, and F1-score. Testing will be conducted on both a held-out validation dataset and real-world email data to ensure the classifier's robustness and generalization capability.

Deployment and Integration: Upon successful validation, the email spam classifier will be deployed and integrated into email server systems or applications, enabling real-time filtering and classification of incoming emails. Integration with existing email platforms will be explored to seamlessly incorporate the classifier into users' email workflows.

1.7 Existing System

1.7.1 Disadvantages of Existing System:

The existing systems for email spam detection often face several limitations and drawbacks, including:

Limited Accuracy: Traditional rule-based spam filters may lack the sophistication to accurately distinguish between spam and legitimate emails, leading to false positives or false negatives.

Static Rule Sets: Rule-based filters rely on predefined sets of rules and heuristics to identify spam patterns, making them vulnerable to evasion tactics employed by spammers who continually adapt their techniques.

Scalability Issues: Some existing systems may struggle to scale efficiently to handle large volumes of email traffic, resulting in performance degradation and latency in email processing.

Inability to Adapt: Static systems may struggle to adapt to emerging spam trends and evolving email threats, requiring frequent manual updates and maintenance to remain effective.

1.7.2 Literature Survey:

A comprehensive literature survey provides insights into the state-of-the-art techniques and approaches used in email spam classification and related fields. Key findings from the literature survey include:

Machine Learning Models: Various machine learning algorithms, including Naive Bayes, Support Vector Machines (SVM), Decision Trees, and Neural Networks, have been employed for email spam classification, with each approach offering unique advantages and trade-offs in terms of accuracy and computational complexity.

Natural Language Processing Techniques: NLP techniques such as tokenization, stopwords removal, stemming, lemmatization, and feature extraction have been extensively used to preprocess textual data and extract relevant features for email spam classification. Advanced NLP models like Word2Vec, GloVe, and BERT have shown promise in capturing semantic relationships and contextual information in text data.

Ensemble Methods: Ensemble learning techniques, such as Random Forests, Gradient Boosting, and Stacking, have been applied to combine multiple base classifiers to improve overall classification performance and robustness against overfitting.

Evaluation Metrics: Commonly used evaluation metrics for assessing the performance of email spam classifiers include accuracy, precision, recall, F1-score, receiver operating characteristic (ROC) curve, and area under the curve (AUC). These metrics provide quantitative measures of the classifier's effectiveness in distinguishing between spam and legitimate emails.

Real-world Applications: Email spam classification has widespread real-world applications in email security, online advertising, and fraud detection. Successful implementation of accurate spam filters can significantly enhance email communication efficiency and user productivity while mitigating security risks associated with spam emails.

By conducting a thorough literature survey, valuable insights and best practices from existing research are gathered, guiding the design and implementation of the email spam classifier in this project.

1.8 Timeline

The timeline for the "Email Spam Classifier using ML Model with NLP Techniques" project outlines the proposed schedule and milestones for its successful completion. It provides a clear overview of the project's timeline, allowing stakeholders to track progress, allocate resources, and ensure timely delivery. The timeline is defined based on a careful analysis of the project's scope, complexity, and the availability of resources.

Phase 1: Project Scope, Planning, and Task Definition (Duration: 1 week)

Project Scope Definition: In this phase, the scope of the project is clearly defined, aligning with the objective of developing an email spam classifier using machine learning (ML) models with natural language processing (NLP) techniques. This includes determining the boundaries of the project, its goals, and what is to be achieved in terms of accurately identifying and classifying spam emails.

Planning: The planning stage involves developing a project plan that outlines the tasks, milestones, timelines, and available resources necessary for the project's successful execution. Given the project's focus on ML and NLP techniques for email spam classification, planning will encompass the selection of appropriate algorithms, data preprocessing methods, and evaluation metrics.

Task Definition: Tasks related to data collection, preprocessing, model development, evaluation, and deployment are defined. The focus is on how these tasks contribute to the project's objective of building an efficient email spam classifier using ML and NLP techniques.

Phase 2: Literature Review (Duration: 2 weeks)

Literature Review: During this phase, a comprehensive review of existing literature related to email spam classification, machine learning, and natural language processing is conducted. The objective is to gather insights and knowledge that will inform the project's design and implementation, supporting the use of advanced techniques in spam detection.

Phase 3: Preliminary Design (Duration: 2 weeks)

Preliminary Design: This phase focuses on developing an initial design for the email spam classifier. It includes defining the architecture and framework for the predictive models that will be used to classify emails as spam or legitimate. The design should consider data sources, feature extraction methods, model selection, and initial experiments with NLP techniques.

Phase 4: Detailed System Design/Technical Details (Duration: 3 weeks)

Detailed System Design: In this phase, the technical details of the email spam classifier are elaborated. This includes specifying the data collection methods, feature extraction techniques, model selection, hyperparameter tuning, and the integration of NLP techniques for text analysis. The design should be aligned with the project's objective of building a robust and accurate classifier.

Phase 5: Work Ethics (Duration: 1 week)

Work Ethics: This phase emphasizes the ethical aspects of the project, ensuring that the work is conducted with integrity, confidentiality, and respect for user privacy. It involves maintaining transparency in the project's activities and adhering to ethical standards in the collection and handling of email data.

Phase 6: End-Term Evaluation (Duration: 2 weeks)

This phase involves evaluating the performance of the email spam classifier based on predefined metrics and criteria. It includes testing the classifier on unseen data, analyzing its accuracy, precision, recall, and F1-score, and making any necessary refinements to improve its performance.

1.9 Organization of our Report:

This project report is structured into the following chapters, providing a comprehensive overview of the "Email Spam Classifier using ML Model with NLP Techniques" project:

Chapter 1: Introduction

In this opening chapter, we introduce the project title, "Email Spam Classifier using ML Model with NLP Techniques," and its central objective—developing an efficient classifier to distinguish between spam and legitimate emails using machine learning (ML) models integrated with natural language processing (NLP) techniques. The chapter sets the context for the entire report, outlining the significance of the project and its relevance in addressing the persistent issue of email spam.

Chapter 2: Literature Review

The literature review chapter delves into the existing body of knowledge related to email spam classification, machine learning, and natural language processing. It explores various approaches, algorithms, and techniques employed in the field, providing a foundation for the project's design and implementation. Additionally, the chapter discusses relevant research studies, benchmark datasets, and best practices in email spam detection, informing the methodology adopted in this project.

Chapter 3: Design and Flow Chart

In this chapter, we present the design and flow chart of the email spam classifier system. The chapter outlines the architecture of the system, including data collection methods, preprocessing steps, feature extraction techniques, model selection, and integration of NLP components. A detailed flow chart illustrates the sequential steps involved in processing and classifying incoming emails, highlighting the key components and decision points in the classification pipeline.

Chapter 4: Result Analysis and Validation

The result analysis and validation chapter assesses the performance of the email spam classifier based on predefined evaluation metrics and criteria. It includes a comprehensive analysis of classification results, model accuracy, precision, recall, and F1-score. Furthermore, the chapter discusses the validation process, including testing the classifier on unseen data and cross-validation techniques to ensure robustness and generalization capability. The findings are critically analyzed and interpreted to evaluate the effectiveness of the classifier in mitigating email spam.

Chapter 5: Conclusion and Recommendations

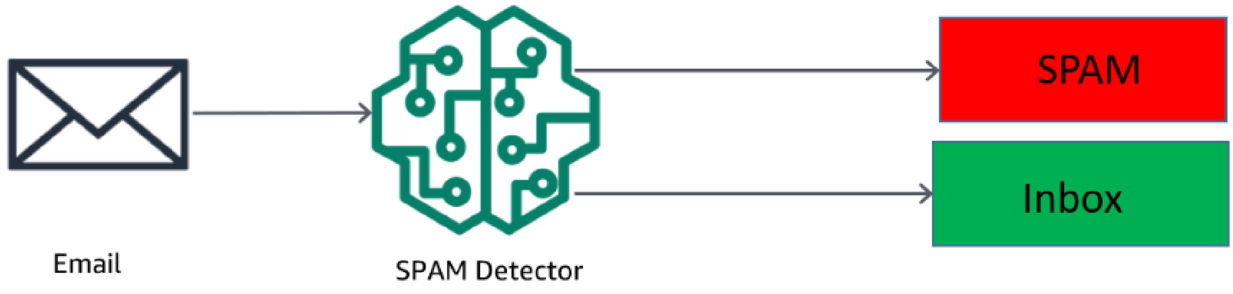
In the concluding chapter, we summarize the key findings and insights derived from the project. Conclusions are drawn regarding the performance of the email spam classifier and its implications for enhancing email security. Additionally, the chapter offers recommendations for further research and improvement, highlighting potential areas for refinement and optimization. Ethical considerations related to data privacy, model fairness, and transparency are also addressed, along with future prospects in the field of email spam detection and cybersecurity.

CHAPTER 2

Project Description

2.1 Introduction

Email communication has become a ubiquitous form of correspondence in both personal and professional spheres. However, alongside its convenience, email also presents the challenge of spam—unsolicited and often malicious messages that clutter inboxes and pose security risks. In response to this challenge, the "Email Spam Classifier using ML Model with NLP Techniques" project aims to develop a sophisticated solution to automatically identify and classify spam emails, thereby enhancing email security and user experience.



Objective: The primary objective of the project is to design, implement, and evaluate an email spam classifier leveraging machine learning (ML) models integrated with natural language processing (NLP) techniques. The classifier will analyze the textual content of incoming emails and determine whether they are spam or legitimate, based on learned patterns and features.

Scope: The scope of the project encompasses several key aspects, including:

Developing a machine learning model capable of accurately distinguishing between spam and legitimate emails.

Preprocessing textual data using NLP techniques to extract relevant features and patterns.

Exploring various ML algorithms and NLP methods to optimize the classifier's performance.

Evaluating the classifier's effectiveness through rigorous testing and validation on real-world email data.

Integrating the classifier into email server systems or applications for real-time spam filtering and classification.

Methodology: The project will follow a systematic methodology consisting of several stages:

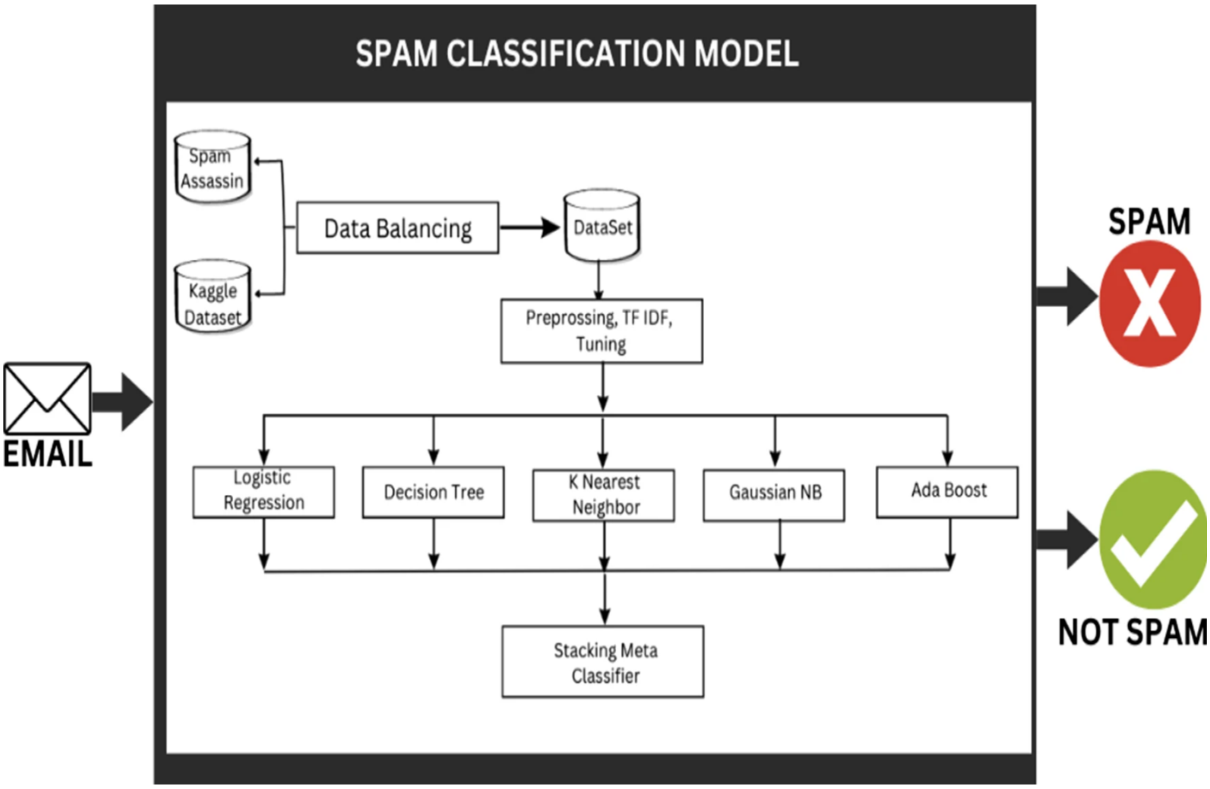


Fig. 4.1: Working of Email Spam Classifier Model

Data Collection and Preprocessing: Acquiring a diverse dataset of labeled emails containing both spam and legitimate examples. Preprocessing the textual data using NLP techniques such as tokenization, stopwords removal, and feature extraction.

Model Development: Experimenting with different ML algorithms, including Naive Bayes, Support Vector Machines (SVM), decision trees, and neural networks, to build the email spam classifier. Exploring advanced techniques such as word embeddings and attention mechanisms to enhance classification accuracy.

Evaluation and Validation: Evaluating the performance of the classifier using standard metrics such as accuracy, precision, recall, and F1-score. Conducting validation tests on held-out datasets and cross-validation to assess robustness and generalization capability.

Deployment and Integration: Integrating the classifier into email server systems or applications to enable real-time spam filtering. Implementing monitoring and alerting mechanisms to notify users of classified spam emails.

Expected Outcomes: The successful completion of the project is expected to yield several outcomes:

A highly accurate and efficient email spam classifier capable of automatically filtering out spam emails.

Improved email security and user experience by reducing the incidence of spam in email inboxes.

Insights into the effectiveness of different ML algorithms and NLP techniques for email spam classification.

Practical implications for integrating ML-based spam filters into email server systems or applications.

In summary, the "Email Spam Classifier using ML Model with NLP Techniques" project addresses a critical need for enhancing email security and user productivity by developing an advanced solution for identifying and classifying spam emails. By leveraging ML models and NLP techniques, the project aims to mitigate the impact of spam and contribute to a safer and more efficient email communication environment.

2.2 Detailed Diagram

2.2.1 Frontend Design

The frontend design of the "Email Spam Classifier using ML Model with NLP Techniques" project encompasses the user interface through which users interact with the classifier system. The frontend design aims to provide a user-friendly and intuitive interface for users to submit emails for classification and view the classification results. The following components are included in the frontend design:

User Interface (UI): The UI consists of web pages or graphical user interfaces (GUIs) where users can input email text or upload email files for classification. The UI may include text input fields, file upload buttons, and interactive elements for submitting emails and viewing classification results.

Input Form: The input form allows users to enter or paste the text of an email that they wish to classify. It may include options for selecting the email source (e.g., manual input, file upload) and additional fields for metadata such as sender information or email subject.

File Upload Feature: Users can upload email files in various formats (e.g., .txt, .eml) for classification. The frontend design includes a file upload feature with drag-and-drop functionality or browse buttons for selecting email files from local storage.

Classification Results Display: Once the email is submitted for classification, the frontend displays the classification results indicating whether the email is classified as spam or legitimate. The results may include additional information such as confidence scores or probability estimates associated with the classification.

Feedback Mechanism: The frontend design may incorporate a feedback mechanism where users can provide feedback on the classification results (e.g., marking misclassified emails, providing additional context). This feedback loop helps improve the accuracy and performance of the classifier over time.

2.2.2 Backend Design

The backend design of the project encompasses the server-side components responsible for processing email data, executing the classification algorithm, and returning classification results to the frontend. The backend design is responsible for handling data storage, model inference, and communication with the frontend. The following components are included in the backend design:

Server Infrastructure: The backend design includes server infrastructure to host the classifier system, handle incoming requests from the frontend, and execute classification tasks. The server may be implemented using cloud-based services or on-premises servers.

API Endpoints: The backend exposes API endpoints that the frontend interacts with to submit email data for classification and retrieve classification results. The API endpoints handle incoming requests, parse input data, and invoke the classification algorithm.

Data Preprocessing: Before classification, the backend preprocesses incoming email data using NLP techniques to tokenize text, remove stopwords, and extract features. Preprocessing steps ensure that the input data is in a suitable format for classification.

Classification Algorithm: The backend executes the classification algorithm, which may be implemented using machine learning models such as Naive Bayes, Support Vector Machines (SVM), or neural networks. The classification algorithm analyzes the preprocessed email data and predicts whether the email is spam or legitimate.

Model Deployment: Trained ML models are deployed on the backend server to perform real-time classification of incoming emails. Model deployment involves loading the trained model into memory and initializing it for inference.

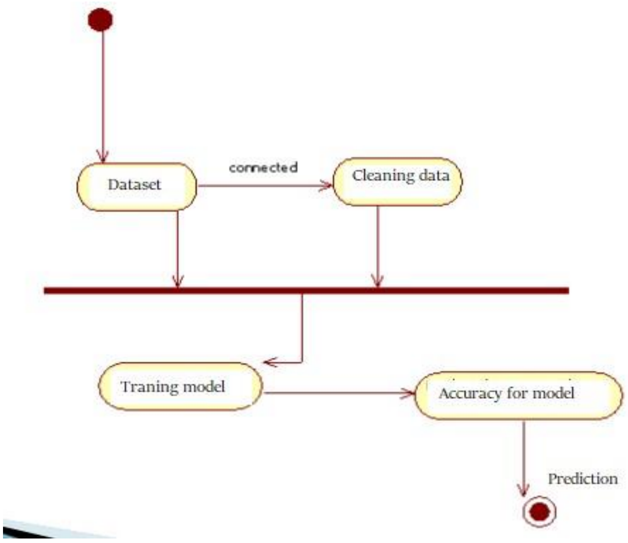
Database Integration: The backend may integrate with a database system to store and retrieve classification results, user feedback data, and other relevant information. The database ensures data persistence and enables tracking of classification history and performance metrics.

Overall, the frontend and backend designs work together seamlessly to provide users with a reliable and efficient email spam classification system. The frontend facilitates user interaction and displays classification results, while the backend handles data processing, model inference, and communication with the frontend. This architecture ensures a robust and scalable solution for classifying emails in real-time.

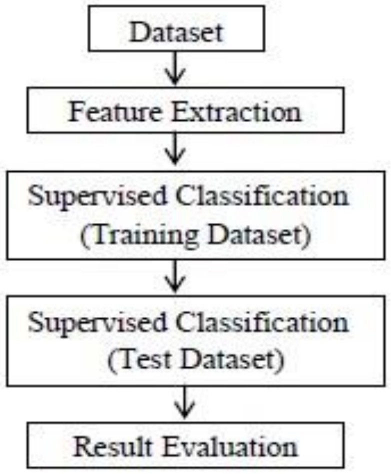
2.3 Module Description

2.3.1 Data Collection

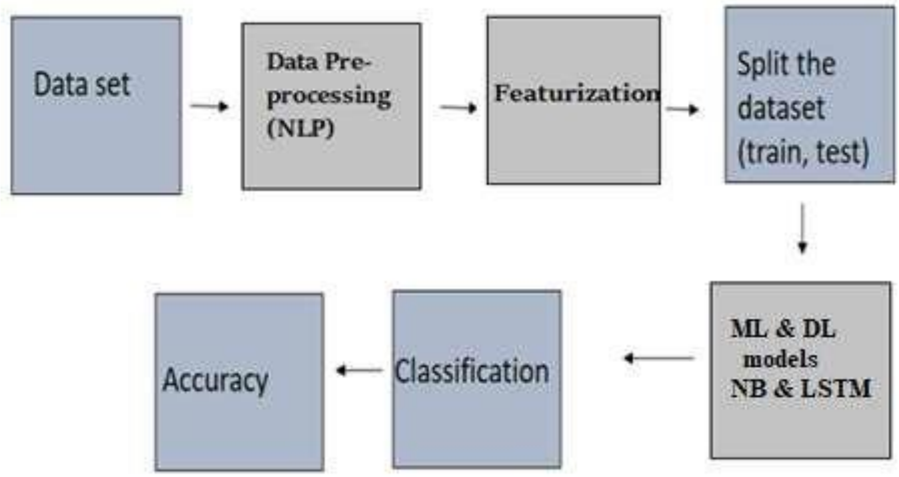
Data Collection module is responsible for acquiring a diverse and representative dataset of emails for training and evaluating the email spam classifier. This module includes the following components:



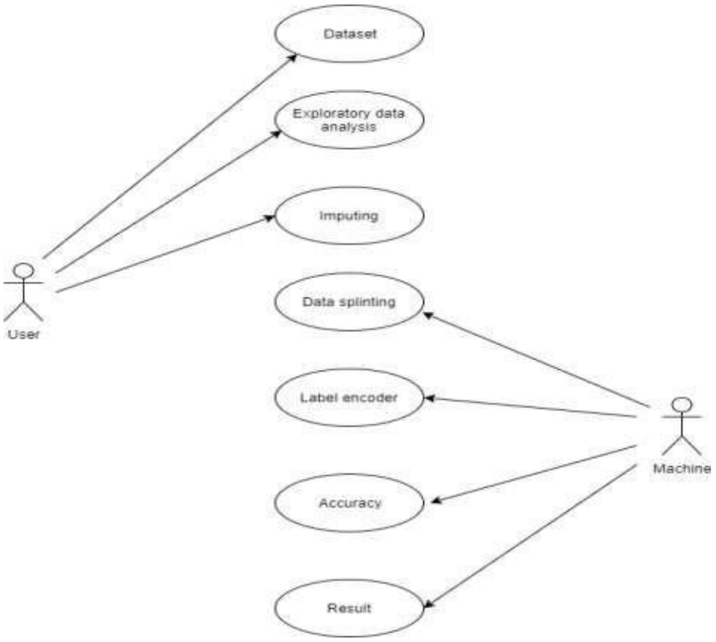
Data Sources: Identifying and accessing sources of email data, including public email corpora, online repositories, and proprietary datasets. Data may be collected from various sources to ensure diversity and relevance to the target domain.


Dataset
Feature Extraction
Supervised Classification (Training Dataset)
Supervised Classification (Test Dataset)
Result Evaluation

Data Crawling and Scraping: Crawling and scraping email data from online sources such as email archives, forums, or social media platforms. This process involves retrieving email content, metadata (e.g., sender, subject), and labels (spam or legitimate) from publicly available sources.



Data Preprocessing: Preprocessing the collected email data to clean and standardize the text, remove duplicates, and handle missing values. Preprocessing steps may include text normalization, tokenization, and encoding email content into a suitable format for further processing.

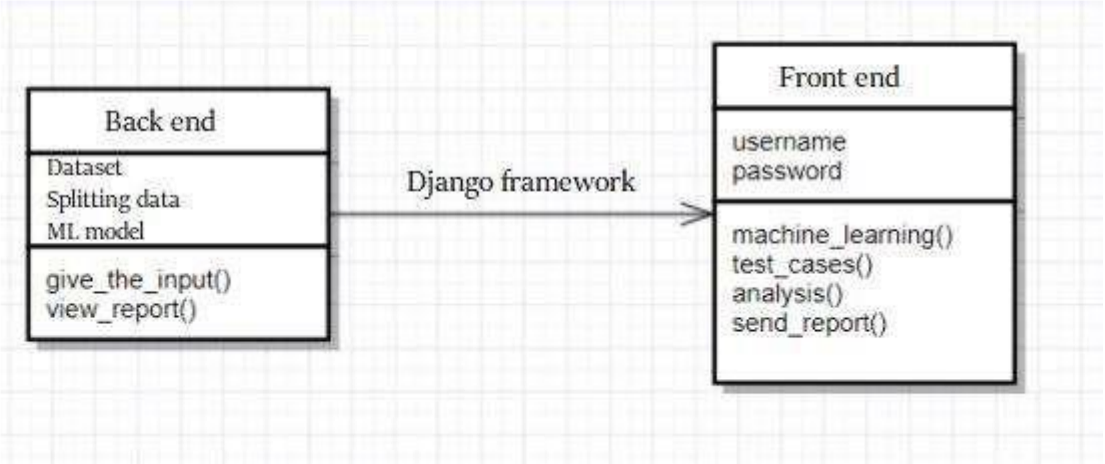


2.3.2 Data Augmentation

Data Augmentation module focuses on enhancing the diversity and quantity of the training dataset by generating synthetic examples of email data. This module includes the following components:

Text Transformation Techniques: Applying text transformation techniques such as synonym replacement, word shuffling, or character-level perturbations to generate variations of existing email samples. These transformations help create new instances of email data while preserving semantic meaning.

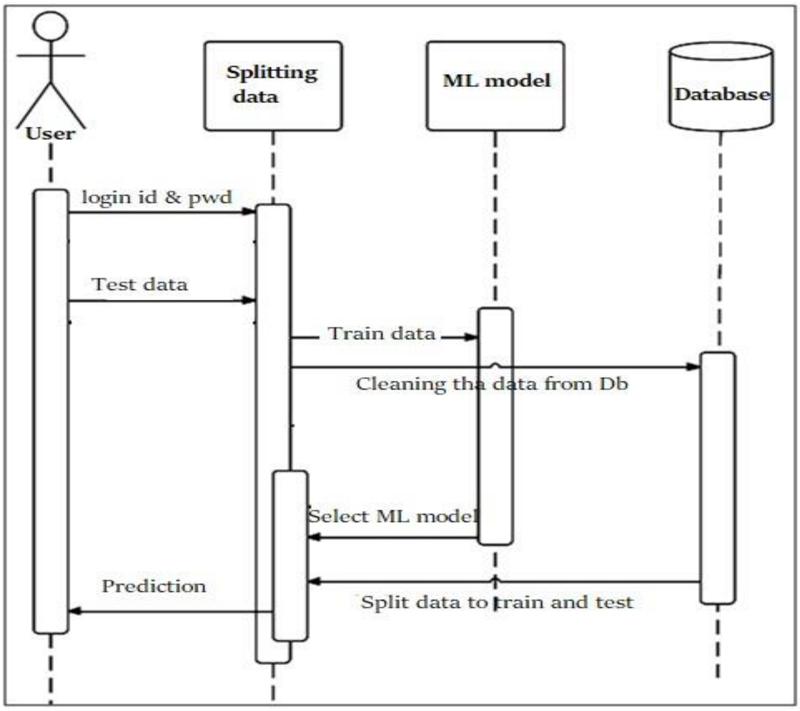
Augmentation Strategies: Designing augmentation strategies to systematically generate diverse examples of spam and legitimate emails. Strategies may include oversampling minority classes, introducing noise or perturbations, or simulating variations in email content.



Quality Control: Ensuring the quality and relevance of augmented data by validating generated examples against predefined criteria. Quality control measures may involve manual inspection, automated checks, or human-in-the-loop validation to maintain data integrity.

2.3.3 Data Splitting

Data Splitting module is responsible for partitioning the dataset into training, validation, and test sets to facilitate model training, hyperparameter tuning, and performance evaluation. This module includes the following components:



Train-Validation-Test Split: Dividing the dataset into three disjoint subsets: training set for model training, validation set for hyperparameter tuning and model selection, and test set for final evaluation of the trained model.

Stratified Sampling: Ensuring that each subset maintains the same class distribution as the original dataset to prevent bias and ensure representative sampling. Stratified sampling techniques are employed to preserve class proportions across subsets.

Cross-Validation: Optionally, performing cross-validation techniques such as k-fold cross-validation to further assess the robustness and generalization capability of the trained model. Cross-validation involves partitioning the dataset into multiple folds and iteratively training and validating the model on different subsets.

2.3.4 Classification

Classification module encompasses the core functionality of the project, where machine learning models are trained and deployed to classify emails as spam or legitimate. This module includes the following components:

Model Training: Training machine learning models using the labeled training data to learn the underlying patterns and features associated with spam and legitimate emails. Various classification algorithms such as Naive Bayes, Support Vector Machines (SVM), decision trees, or neural networks may be employed.

Hyperparameter Tuning: Optimizing the hyperparameters of the classification models to improve performance and generalization. Hyperparameter tuning techniques such as grid search, random search, or Bayesian optimization are applied to find the optimal configuration.

Model Evaluation: Evaluating the performance of the trained models using evaluation metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve. Model evaluation is conducted on the validation and test sets to assess the classifier's effectiveness in distinguishing between spam and legitimate emails.

2.3.5 Performance Metrices

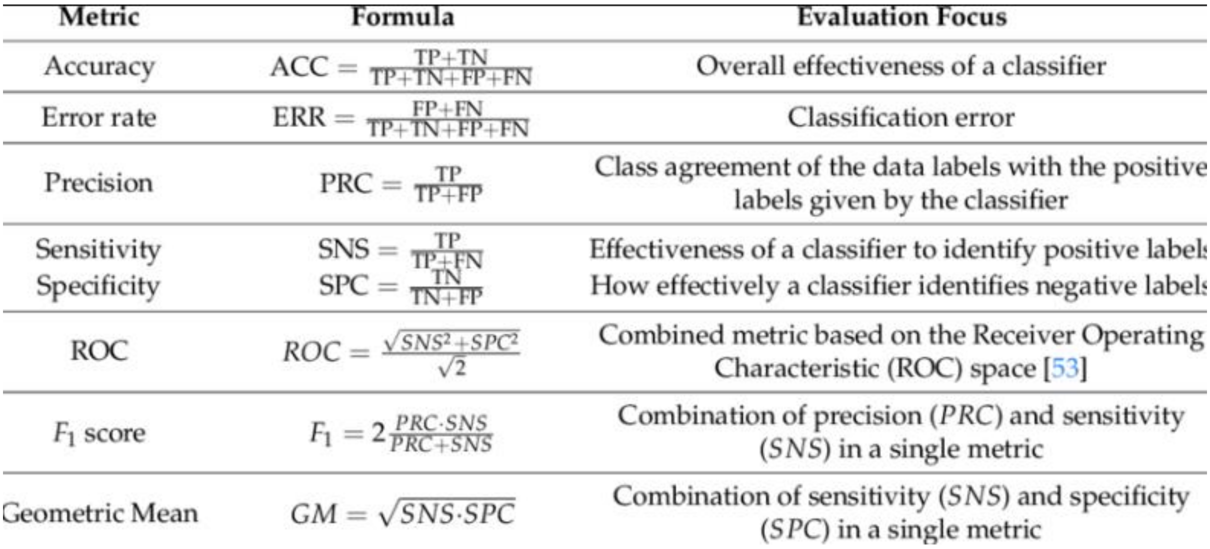
Performance Metrics module is responsible for quantitatively assessing the performance of the email spam classifier using predefined evaluation metrics. This module includes the following components:

Accuracy: The proportion of correctly classified emails out of the total number of emails.

Precision: The proportion of true positive predictions out of all positive predictions, indicating the classifier's ability to avoid false positives.

Recall (Sensitivity): The proportion of true positive predictions out of all actual positive instances, indicating the classifier's ability to capture all positive instances.

F1-score: The harmonic mean of precision and recall, providing a balanced measure of the classifier's performance.

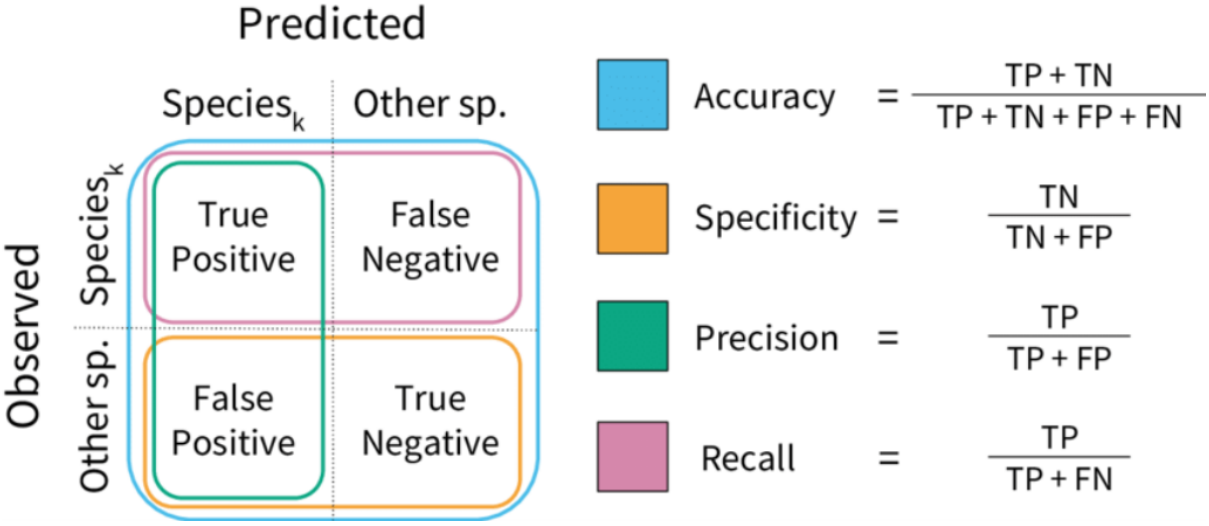


Metric	Formula	Evaluation Focus
Accuracy	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$	Overall effectiveness of a classifier
Error rate	$ERR = \frac{FP+FN}{TP+TN+FP+FN}$	Classification error
Precision	$PRC = \frac{TP}{TP+FP}$	Class agreement of the data labels with the positive labels given by the classifier
Sensitivity Specficity	$SNS = \frac{TP}{TP+FN}$ $SPC = \frac{TN}{TN+FP}$	Effectiveness of a classifier to identify positive labels How effectively a classifier identifies negative labels
ROC	$ROC = \frac{\sqrt{SNS^2+SPC^2}}{\sqrt{2}}$	Combined metric based on the Receiver Operating Characteristic (ROC) space [53]
F_1 score	$F_1 = 2\frac{PRC \cdot SNS}{PRC+SNS}$	Combination of precision (PRC) and sensitivity (SNS) in a single metric
Geometric Mean	$GM = \sqrt{SNS \cdot SPC}$	Combination of sensitivity (SNS) and specificity (SPC) in a single metric

Area under the ROC curve (AUC): The area under the receiver operating characteristic (ROC) curve, representing the classifier's ability to discriminate between spam and legitimate emails across different threshold values.

2.3.6 Confusion Matrix

Confusion Matrix module provides a tabular representation of the classifier's performance by comparing predicted labels with actual labels. This module includes the following components:


Predicted
Species$_k$
Other sp.
Observed
Species$_k$
Other sp.
True Positive
False Negative
False Positive
True Negative
Accuracy = $\frac{TP + TN}{TP + TN + FP + FN}$
Specificity = $\frac{TN}{TN + FP}$
Precision = $\frac{TP}{TP + FP}$
Recall = $\frac{TP}{TP + FN}$

True Positive (TP): Emails correctly classified as spam.

False Positive (FP): Legitimate emails incorrectly classified as spam (Type I error).

True Negative (TN): Legitimate emails correctly classified as legitimate.

False Negative (FN): Spam emails incorrectly classified as legitimate (Type II error).

The confusion matrix allows for a detailed analysis of the classifier's performance, including measures such as precision, recall, specificity, and accuracy.

2.4 Software Specifications

2.4.1 Hardware Specifications

Determining the hardware specifications for the "Email Spam Classifier using ML Model with NLP Techniques" project involves considering factors such as the scale of deployment, computational requirements for training and inference, and budget constraints. While the specific hardware needs may vary based on these factors, a basic setup for development and testing purposes typically includes:

Processor: A multi-core processor is essential for efficient data processing and model training. Processors like Intel Core i5 or AMD Ryzen 5, or higher, are suitable choices. These processors provide the necessary computing power to handle the computational tasks involved in training and testing the machine learning models.

Memory (RAM): Adequate RAM is crucial for handling large datasets and performing memory-intensive operations during model training. A minimum of 8 GB of RAM is recommended to ensure smooth execution of the machine learning algorithms. However, for larger datasets and more complex models, higher RAM capacity, such as 16 GB or 32 GB, may be necessary to prevent performance bottlenecks.

Storage: Sufficient storage space is needed to store datasets, trained models, and other project files. Both solid-state drives (SSDs) and hard disk drives (HDDs) can be used for storage, with SSDs offering faster read/write speeds and better performance. The storage capacity required depends on the size of the datasets and the frequency of data updates. It's advisable to have several hundred gigabytes of storage space available.

Graphics Processing Unit (GPU): While a GPU is optional for development and testing, it can significantly accelerate model training and inference, especially for deep learning models. NVIDIA GPUs, such as those from the GeForce GTX or RTX series, are commonly used for machine learning tasks due to their parallel processing capabilities. If your project involves training large-scale or complex models, investing in a GPU can greatly reduce training time and improve productivity.

For production deployment, a scalable hardware infrastructure is necessary to accommodate varying workloads and ensure high availability and performance. Cloud computing instances, such as AWS EC2 or Google Cloud VMs, offer flexible and scalable resources that can be tailored to the specific requirements of the project. By leveraging cloud-based solutions, you can easily scale your infrastructure as needed, optimize costs, and ensure reliable performance for your email spam classifier application.

2.4.2 Software Specifications

The software specifications required for the project encompass the development environment, libraries, frameworks, and tools necessary for implementing the email spam classifier system. The software stack may include:

Programming Languages: Python is the primary programming language used for implementing machine learning models, data preprocessing, and system development. Libraries such as NumPy, Pandas, and scikit-learn provide essential functionalities for data manipulation, feature extraction, and model training.

Machine Learning Frameworks: TensorFlow and PyTorch are popular deep learning frameworks used for building and training neural network models. These frameworks offer high-level APIs and prebuilt modules for developing complex architectures and conducting experiments efficiently.

Natural Language Processing (NLP) Libraries: NLTK (Natural Language Toolkit), spaCy, and Gensim are NLP libraries used for text preprocessing, tokenization, and feature extraction. These libraries provide tools for handling textual data and implementing advanced NLP techniques such as word embeddings and entity recognition.

Web Development Frameworks: Flask or Django may be used for building the frontend and backend components of the classifier system. Flask is a lightweight and flexible web framework suitable for developing RESTful APIs, while Django provides a full-featured framework for building web applications with scalability and security features.

Database Management Systems: SQLite, PostgreSQL, or MongoDB can be used for storing and managing data related to email classification results, user feedback, and model configurations. These databases offer reliability, scalability, and ACID compliance for data storage and retrieval.

Development Tools: Integrated Development Environments (IDEs) such as PyCharm, Jupyter Notebook, or Visual Studio Code are commonly used for writing, debugging, and testing code. Version control systems like Git facilitate collaboration and code management across team members.

Containerization Tools: Docker and Kubernetes may be used for containerizing and orchestrating the deployment of the classifier system. Containers provide a lightweight and consistent environment for running applications across different platforms, while Kubernetes automates the deployment, scaling, and management of containerized applications in a distributed environment.

2.5 Module Diagram

The module diagram for the email spam classifier system offers a visual representation of its high-level architecture and the interactions between different modules or components. It serves as a roadmap for understanding the system's structure, functionality, and data flow, aiding in system design, development, and maintenance. Here's a detailed breakdown of the components typically included in such a diagram:

Data Collection Module:

This module is responsible for acquiring email data from various sources, such as email servers or datasets.

It includes functionalities for data preprocessing, which may involve tasks like text normalization, tokenization, and removing stop words.

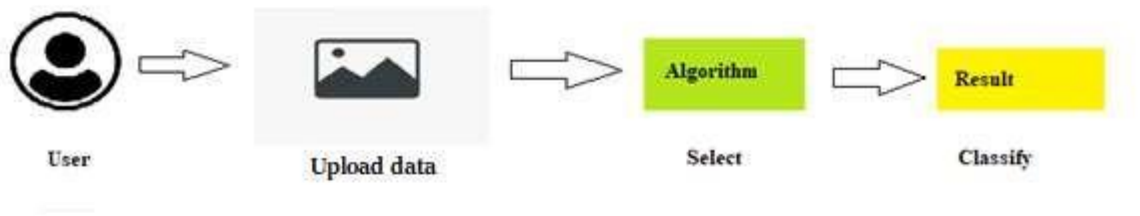
Once the data is processed, it is stored in a database for further analysis and model training.

Data Augmentation Module:

The data augmentation module generates synthetic examples of email data to augment the training dataset.

By creating additional training examples, this module helps improve the robustness and generalization of the machine learning models.

Techniques such as text augmentation, where the original text is modified to create new examples, can be implemented in this module.



Data Splitting Module:

This module splits the dataset into training, validation, and test sets for model training, evaluation, and testing purposes.

It ensures that the model is trained on a subset of the data, validated on another subset to tune hyperparameters, and finally tested on a separate subset to assess its performance.

Classification Module:

The classification module implements machine learning models for email spam classification.

It includes functionalities for preprocessing the email data, extracting relevant features, training the model, and performing inference on new data.

Common techniques used in this module include vectorization of text data, feature extraction using NLP techniques, and training classifiers such as Naive Bayes, SVM, or deep learning models.

Performance Metrics Module:

This module computes evaluation metrics such as accuracy, precision, recall, and F1-score to assess the performance of the classifier.

It evaluates how well the model is able to classify emails as spam or non-spam and provides insights into its strengths and weaknesses.

Confusion Matrix Module:

The confusion matrix module generates a confusion matrix to visualize the classifier's performance and analyze prediction errors.

It helps identify the number of true positives, true negatives, false positives, and false negatives, allowing stakeholders to understand where the classifier is making mistakes.

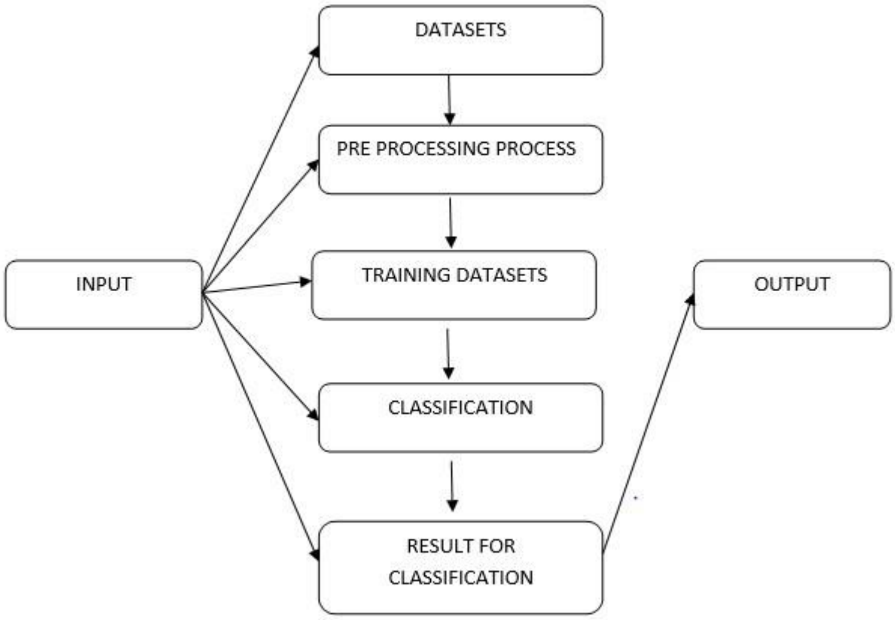


Fig 1.3: Flowchart of Training data and getting the output

By incorporating these components into the module diagram, stakeholders gain a comprehensive understanding of the email spam classifier system's architecture, functionality, and dependencies. This visual representation facilitates collaboration among team members, streamlines development processes, and aids in troubleshooting and maintenance activities.

CHAPTER 3

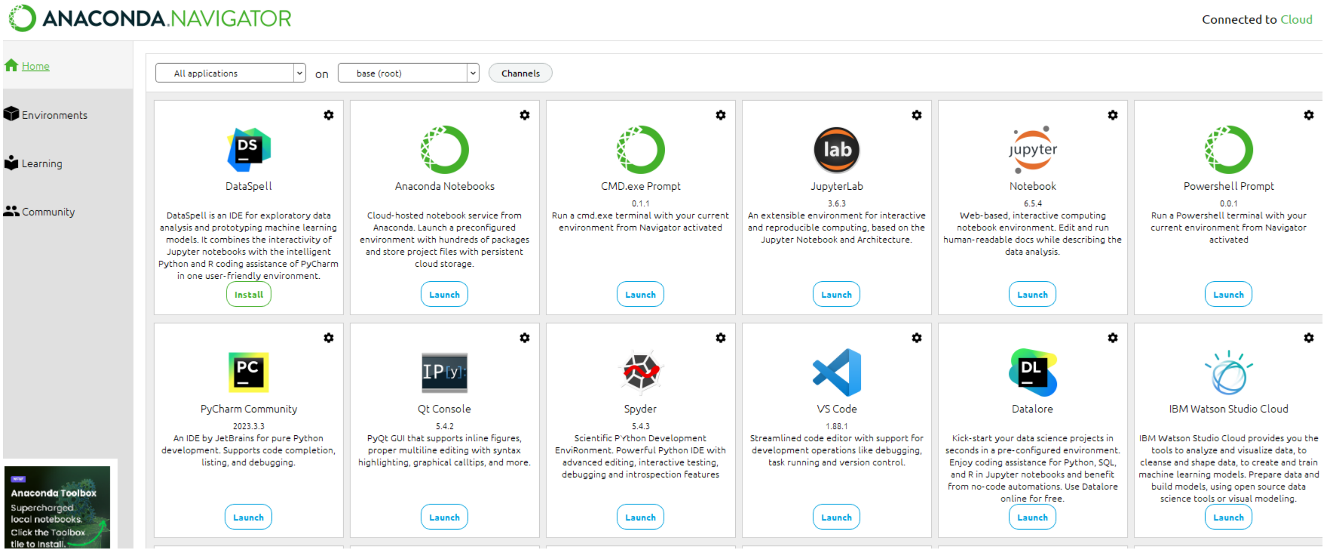
DESIGN FLOW/PROCESS

3.1 General

The software specification for the "Email Spam Classifier using ML Model with NLP Techniques" project outlines the tools, frameworks, and programming languages utilized in the development and implementation of the classifier system. This chapter provides an overview of the general software environment and specific software components relevant to the project.

3.2 Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing, data analysis, and machine learning. Anaconda provides a comprehensive suite of tools, libraries, and packages pre-installed and configured for data science and machine learning tasks. Key features of Anaconda include:



DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Anaconda Notebooks

Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

CMD.exe Prompt
0.1.1

Run a cmd.exe terminal with your current environment from Navigator activated

JupyterLab
3.6.3

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.5.4

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1

Run a Powershell terminal with your current environment from Navigator activated

PyCharm Community
2023.3.3

An IDE by JetBrains for pure Python development. Supports code completion, listing, and debugging.

Qt Console
5.4.2

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
5.4.3

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code
1.88.1

Streamlined code editor with support for development operations like debugging, task running and version control.

Datalore

Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Package Management: Anaconda includes the Conda package manager, which allows users to easily install, update, and manage libraries and dependencies required for data analysis and machine learning projects.

Environment Management: Anaconda enables the creation of isolated Python environments, allowing users to manage dependencies and package versions for different projects without conflicts.

Integrated Development Environment (IDE): Anaconda Navigator provides a graphical user interface (GUI) for managing environments, launching applications, and accessing documentation, tutorials, and sample code.

Comprehensive Libraries: Anaconda comes pre-installed with popular data science libraries such as NumPy, Pandas, Matplotlib, scikit-learn, TensorFlow, and PyTorch, among others, making it suitable for a wide range of machine learning and NLP tasks.

Anaconda simplifies the setup and configuration of the development environment for the project, ensuring consistency and compatibility across different platforms and environments.

3.3 Python

Python is the primary programming language used for implementing the email spam classifier system. Python offers several advantages for machine learning and NLP tasks, including readability, versatility, and a rich ecosystem of libraries and frameworks.

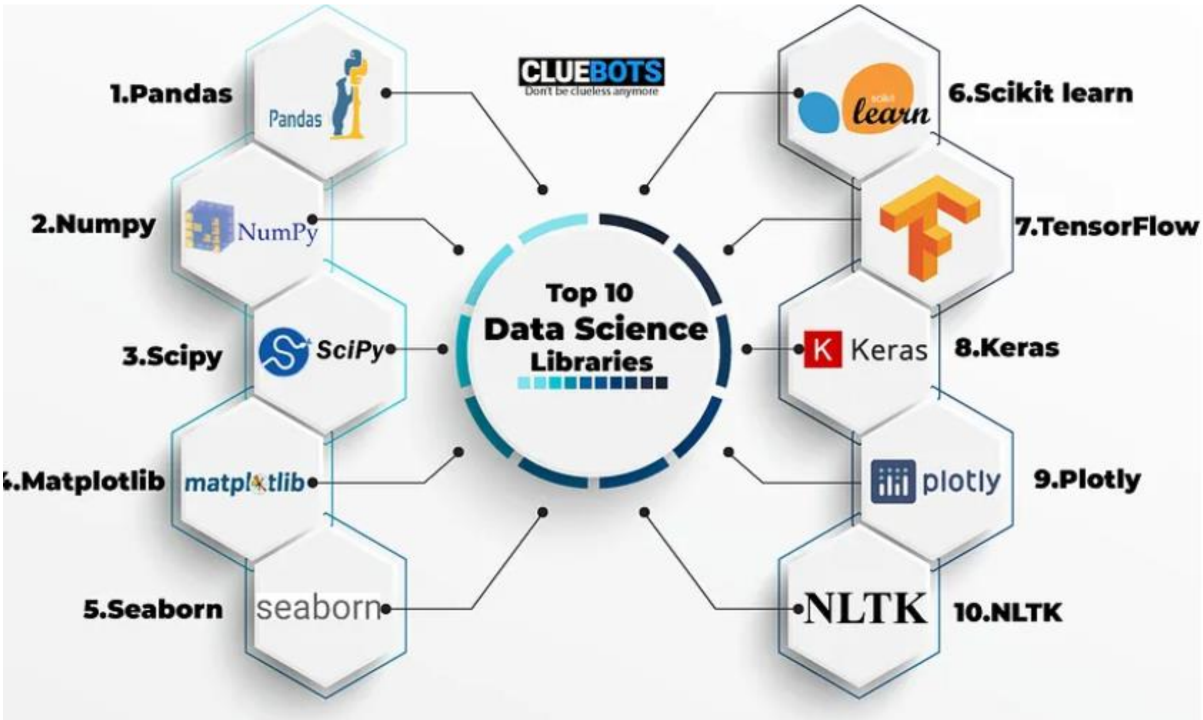
3.3.1 Calculations and Libraries

Python provides robust support for numerical computations, data manipulation, and statistical analysis through libraries such as NumPy and Pandas. NumPy offers efficient data structures and functions for numerical computations, while Pandas provides high-level data structures and tools for data manipulation and analysis.

3.3.2 Python libraries and packages

In addition to NumPy and Pandas, Python boasts a rich ecosystem of libraries and packages tailored for machine learning, natural language processing, and web development. Key libraries and packages relevant to the project include:

Fig 1.4: Python Libraries



scikit-learn: A comprehensive machine learning library that provides implementations of various classification algorithms, preprocessing techniques, and evaluation metrics.

NLTK (Natural Language Toolkit): A popular library for NLP tasks, including tokenization, stemming, lemmatization, part-of-speech tagging, and named entity recognition.

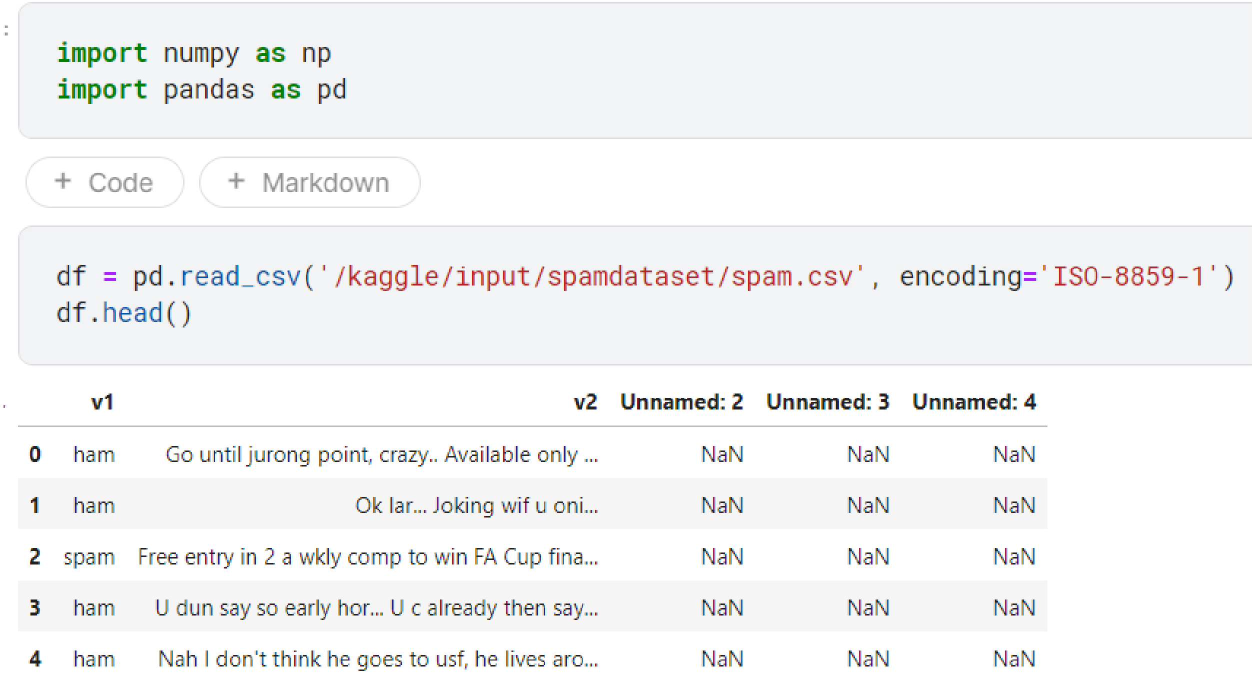
spaCy: Another NLP library offering advanced features such as dependency parsing, entity recognition, and text classification, with a focus on performance and ease of use.

Flask or Django: Web development frameworks used for building the frontend and backend components of the classifier system. Flask is a lightweight and flexible framework suitable for RESTful APIs, while Django provides a full-featured framework for developing web applications with scalability and security features.

CHAPTER 4

RESULTS ANALYSIS AND VALIDATION

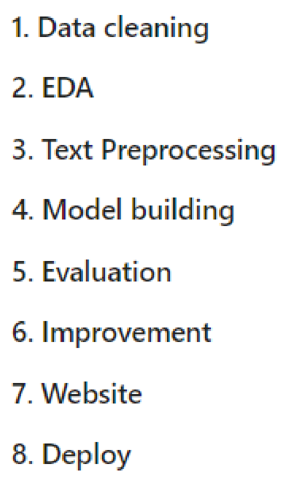
4.1 Implementation



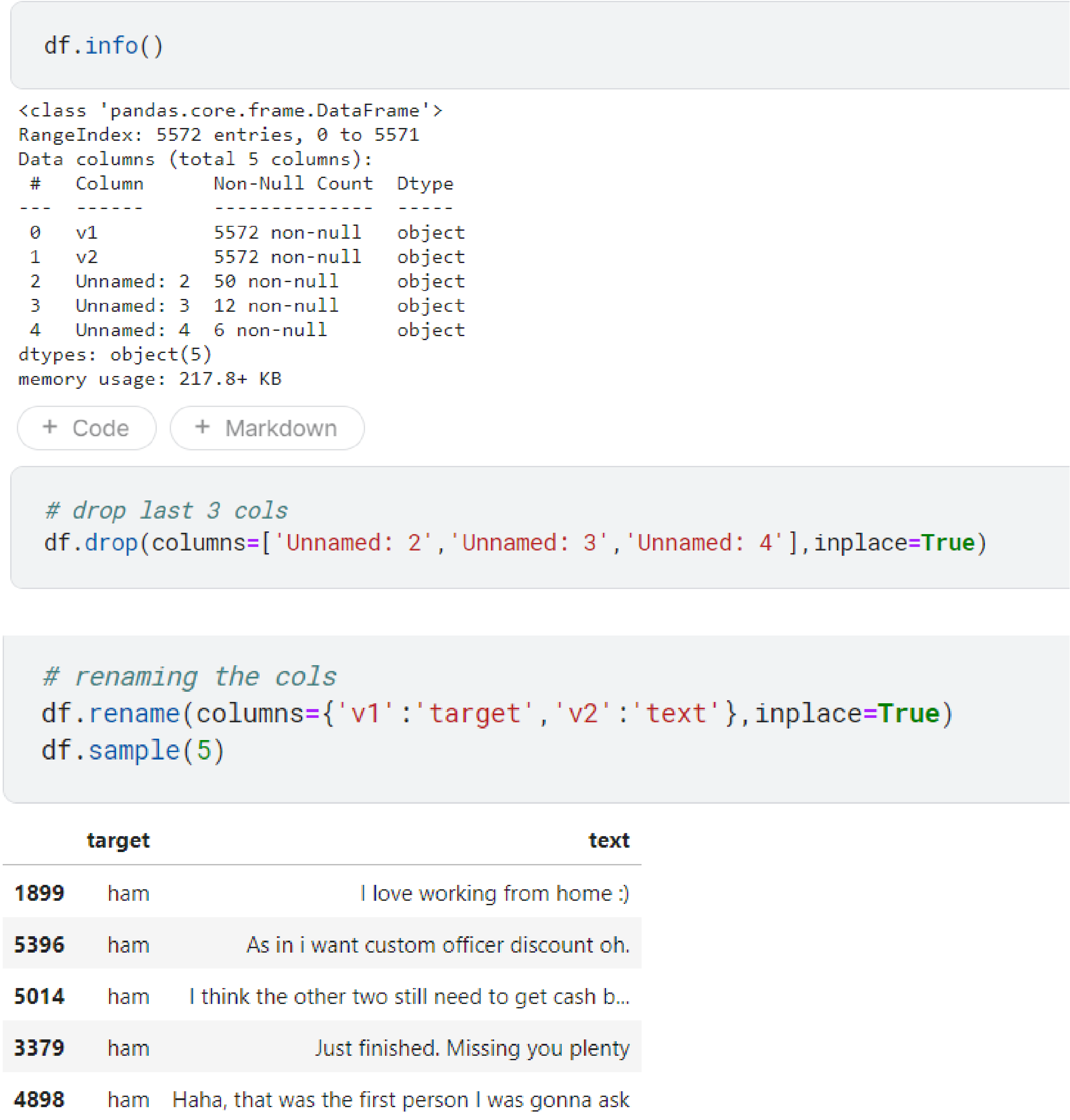
```
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('/kaggle/input/spamdataset/spam.csv', encoding='ISO-8859-1')
df.head()
```

	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



1. Data cleaning
2. EDA
3. Text Preprocessing
4. Model building
5. Evaluation
6. Improvement
7. Website
8. Deploy



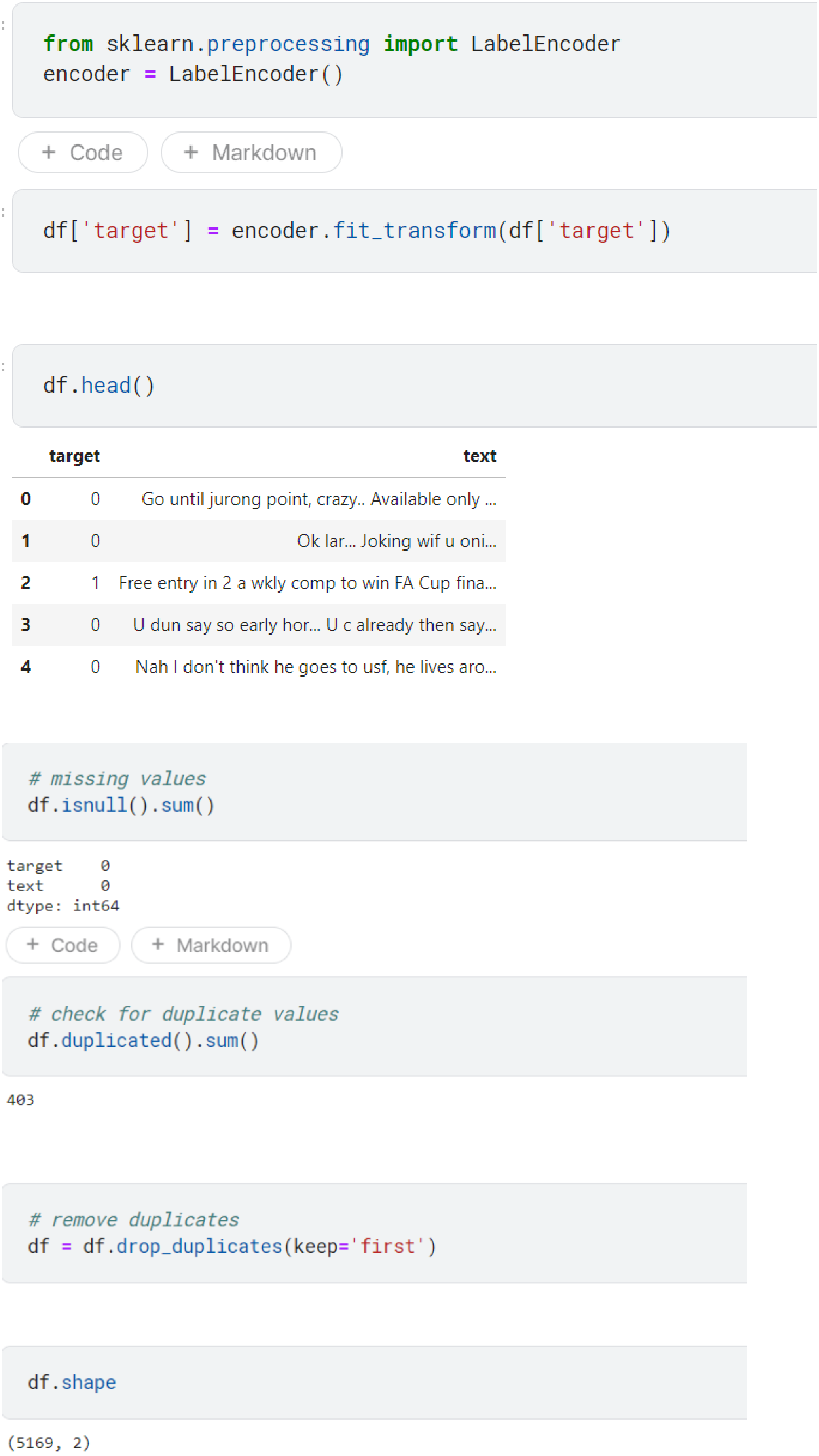
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

```
# drop last 3 cols
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

```
# renaming the cols
df.rename(columns={'v1':'target','v2':'text'},inplace=True)
df.sample(5)
```

	target	text
1899	ham	I love working from home :)
5396	ham	As in i want custom officer discount oh.
5014	ham	I think the other two still need to get cash b...
3379	ham	Just finished. Missing you plenty
4898	ham	Haha, that was the first person I was gonna ask



```python
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

```python
df['target'] = encoder.fit_transform(df['target'])
```

```python
df.head()
```

	target	text
0	0	Go until jurong point, crazy.. Available only ...
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, he lives aro...

```python
# missing values
df.isnull().sum()
```

```
target    0
text      0
dtype: int64
```

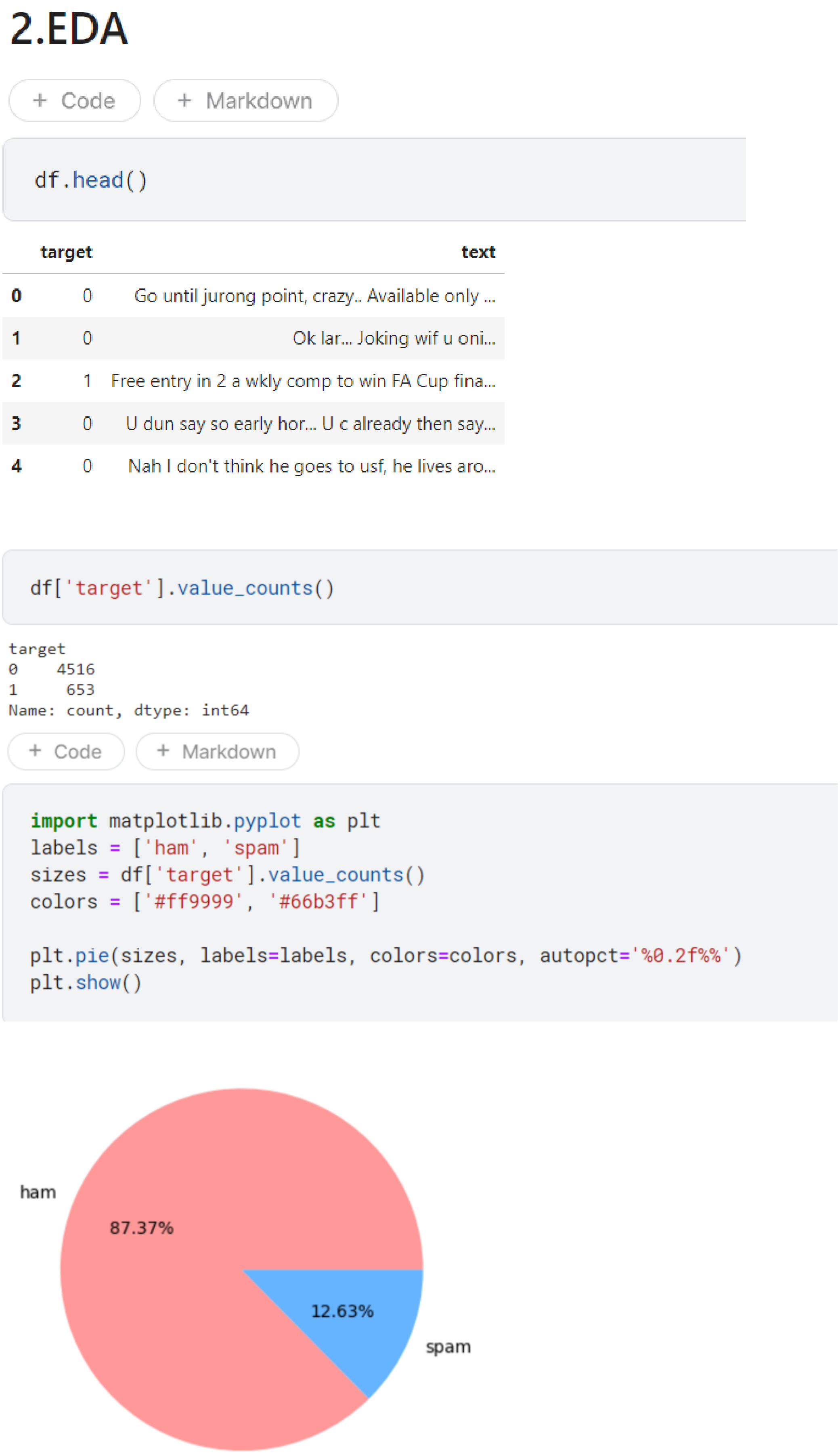
```python
# check for duplicate values
df.duplicated().sum()
```

```
403
```

```python
# remove duplicates
df = df.drop_duplicates(keep='first')
```

```python
df.shape
```

```
(5169, 2)
```



2.EDA

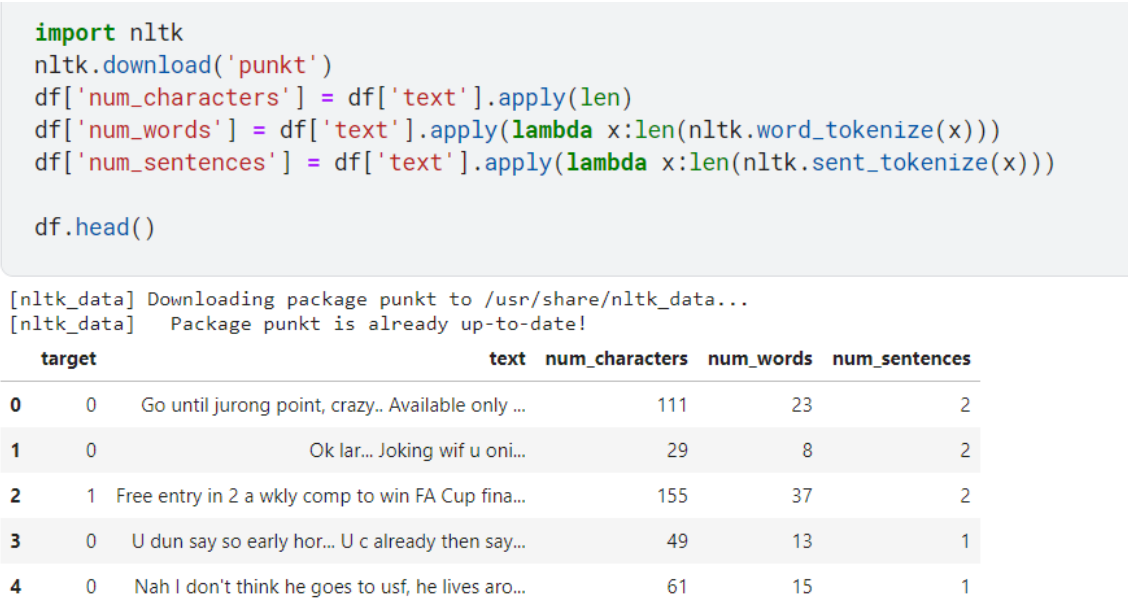
```
df.head()
```

	target	text
0	0	Go until jurong point, crazy.. Available only ...
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, he lives aro...

```
df['target'].value_counts()
```

```
target
0    4516
1     653
Name: count, dtype: int64
```

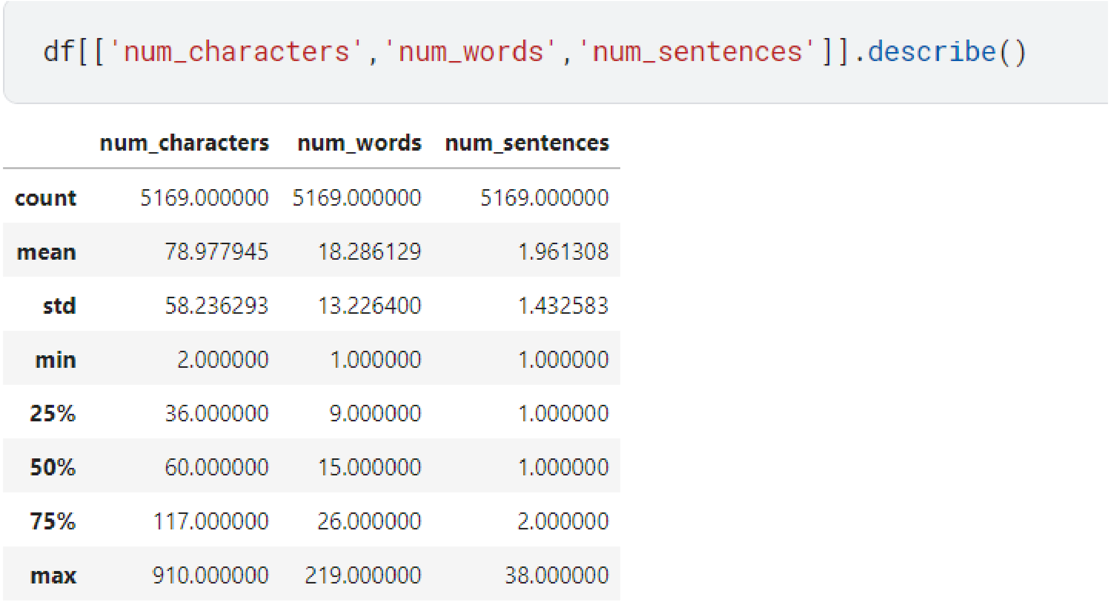
```
import matplotlib.pyplot as plt
labels = ['ham', 'spam']
sizes = df['target'].value_counts()
colors = ['#ff9999', '#66b3ff']

plt.pie(sizes, labels=labels, colors=colors, autopct='%0.2f%%')
plt.show()
```



```python
import nltk
nltk.download('punkt')
df['num_characters'] = df['text'].apply(len)
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
df['num_sentences'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))

df.head()
```

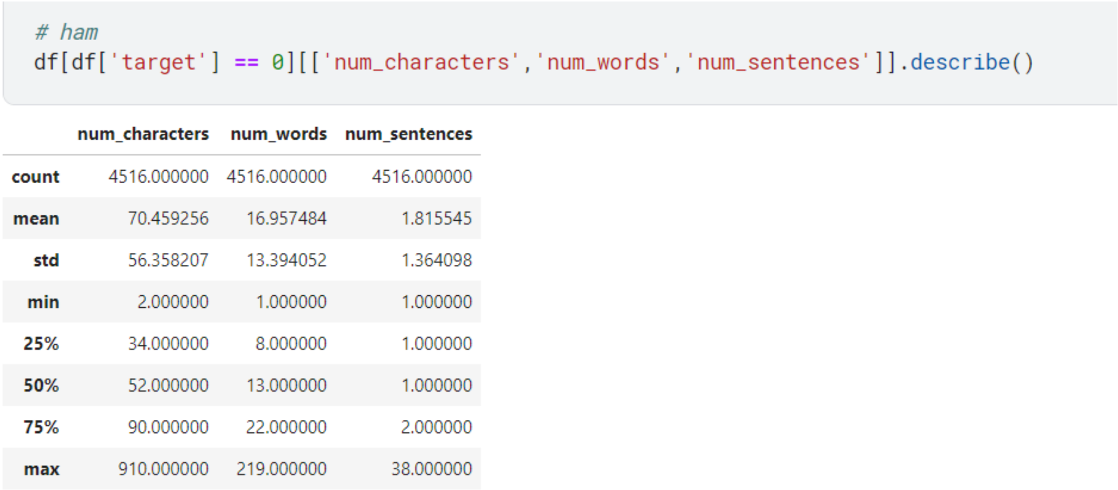
```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

	target	text	num_characters	num_words	num_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	23	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	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, he lives aro...	61	15	1



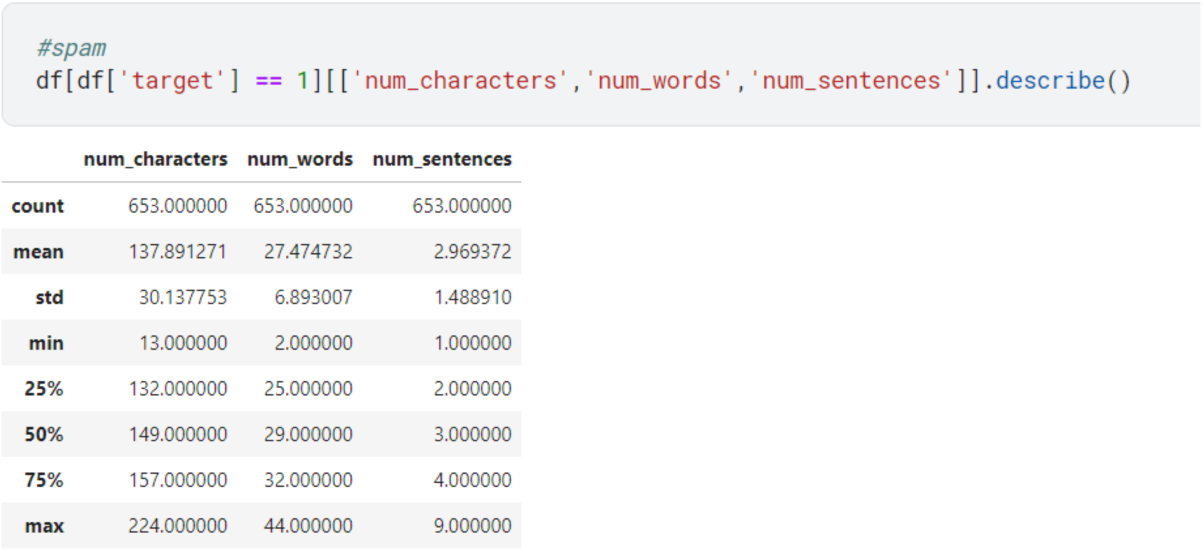
```python
df[['num_characters','num_words','num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.286129	1.961308
std	58.236293	13.226400	1.432583
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000
50%	60.000000	15.000000	1.000000
75%	117.000000	26.000000	2.000000
max	910.000000	219.000000	38.000000



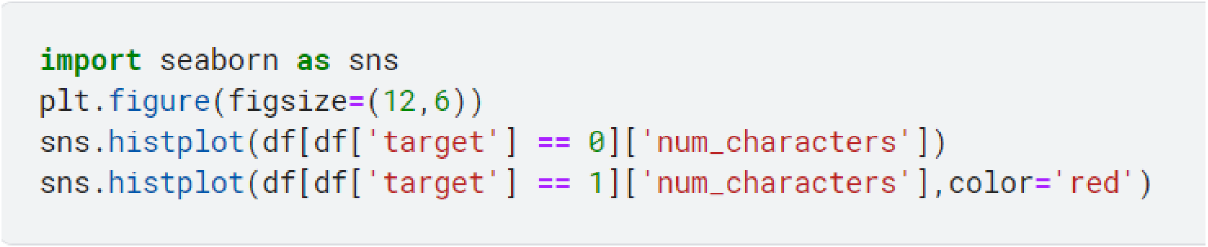
```python
# ham
df[df['target'] == 0][['num_characters','num_words','num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	4516.000000	4516.000000	4516.000000
mean	70.459256	16.957484	1.815545
std	56.358207	13.394052	1.364098
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	219.000000	38.000000

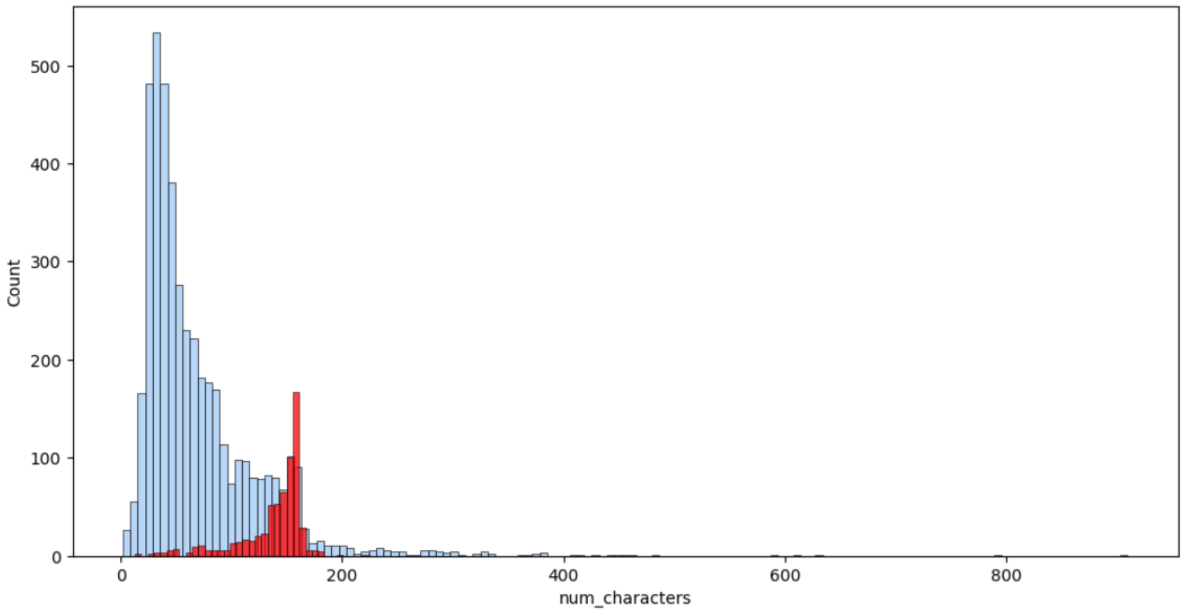


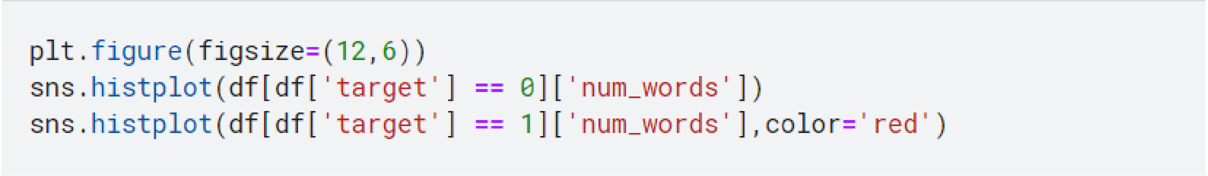
```python
#spam
df[df['target'] == 1][['num_characters','num_words','num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	653.000000	653.000000	653.000000
mean	137.891271	27.474732	2.969372
std	30.137753	6.893007	1.488910
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	44.000000	9.000000

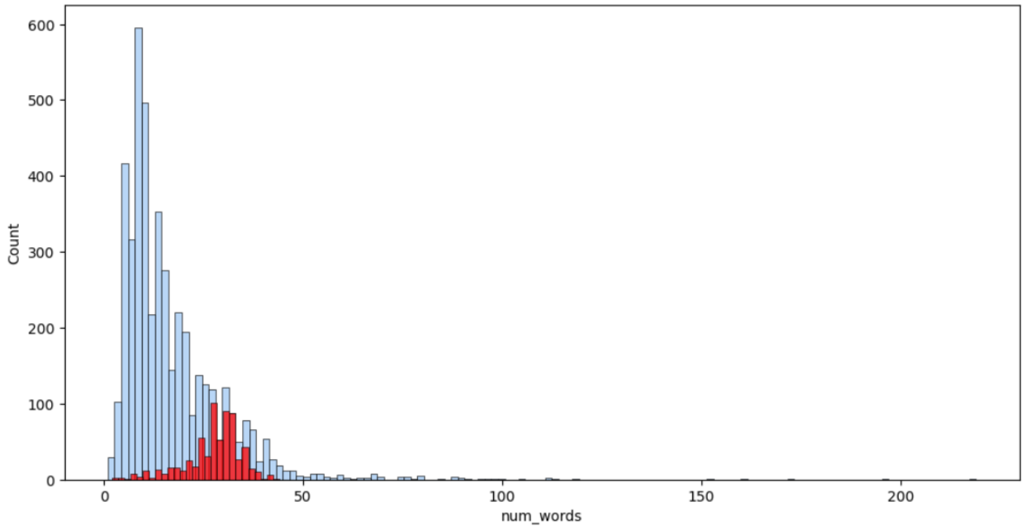


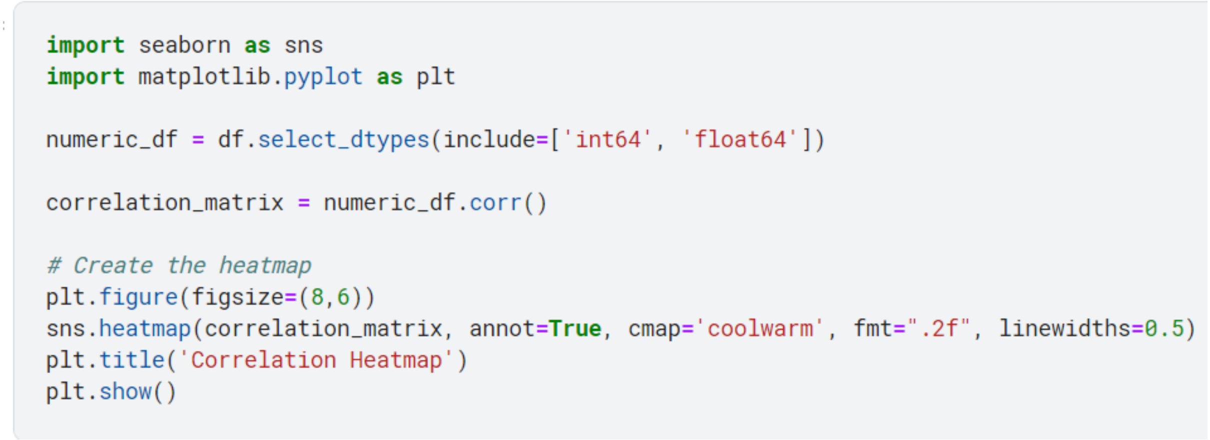
```python
import seaborn as sns
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_characters'])
sns.histplot(df[df['target'] == 1]['num_characters'],color='red')
```



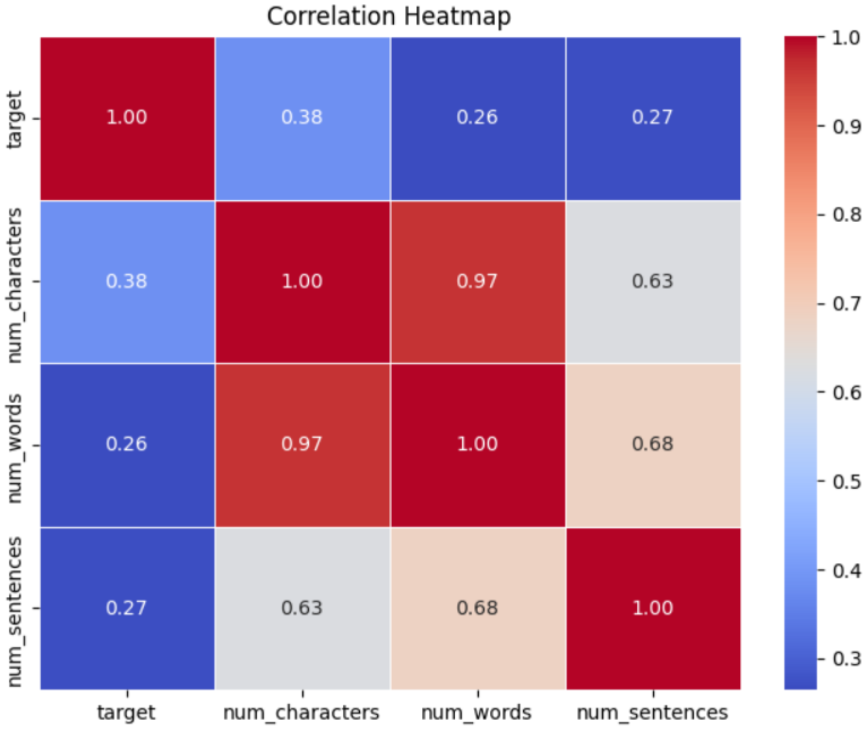


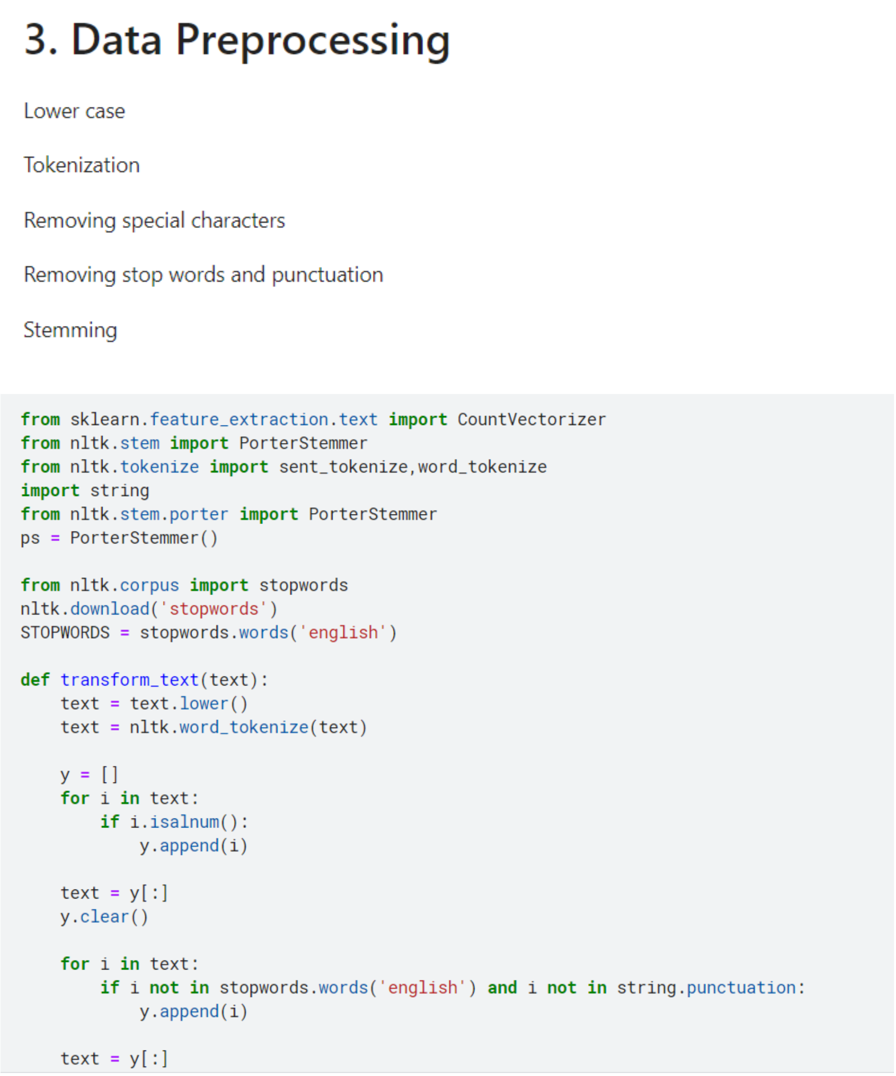
```python
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_words'])
sns.histplot(df[df['target'] == 1]['num_words'],color='red')
```





```
import seaborn as sns
import matplotlib.pyplot as plt

numeric_df = df.select_dtypes(include=['int64', 'float64'])

correlation_matrix = numeric_df.corr()

# Create the heatmap
plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```





3. Data Preprocessing

Lower case

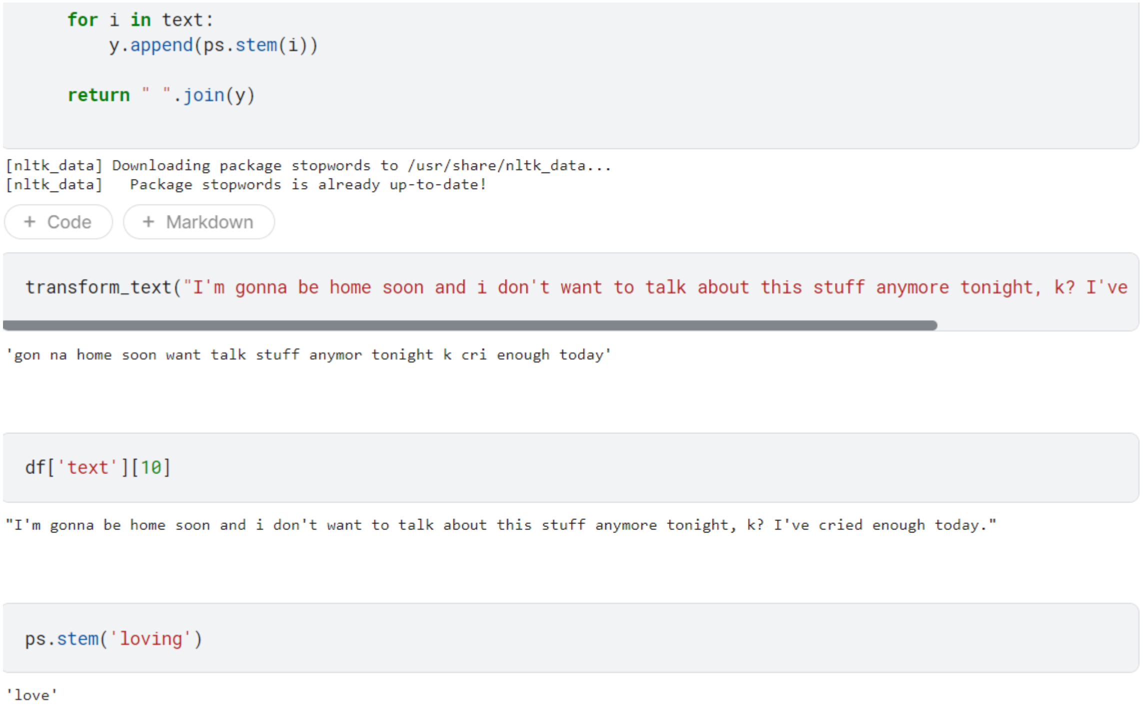
Tokenization

Removing special characters

Removing stop words and punctuation

Stemming

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize,word_tokenize
import string
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

from nltk.corpus import stopwords
nltk.download('stopwords')
STOPWORDS = stopwords.words('english')

def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)

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

    text = y[:]
    y.clear()

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

    text = y[:]
```



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

    return " ".join(y)
```

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

```
transform_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've
```

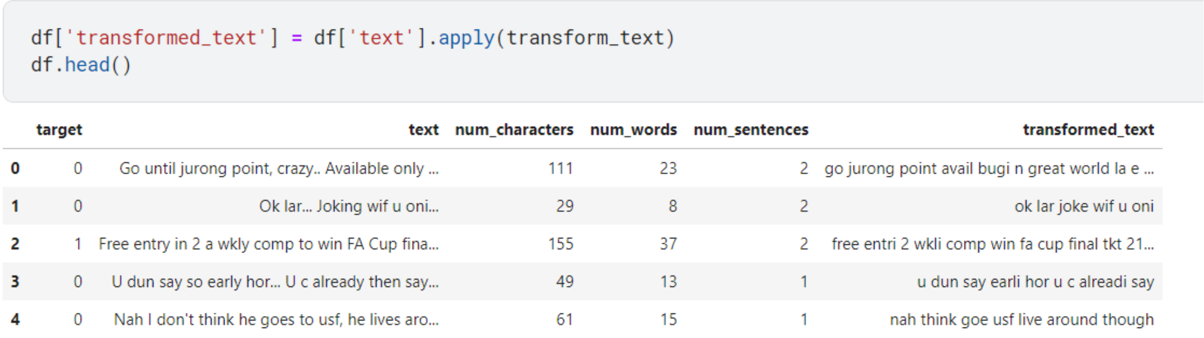
```
'gon na home soon want talk stuff anymor tonight k cri enough today'
```

```
df['text'][10]
```

```
"I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today."
```

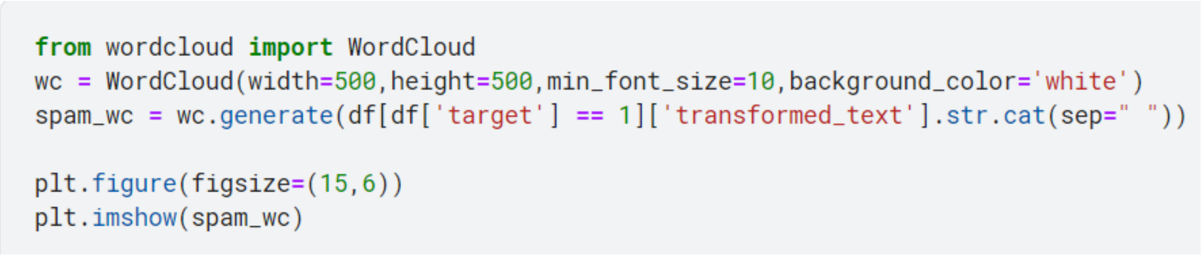
```
ps.stem('loving')
```

```
'love'
```

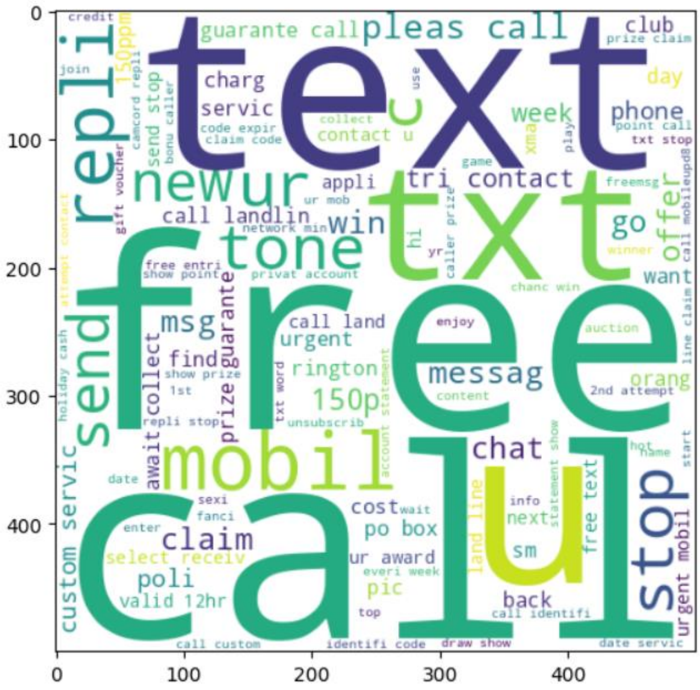


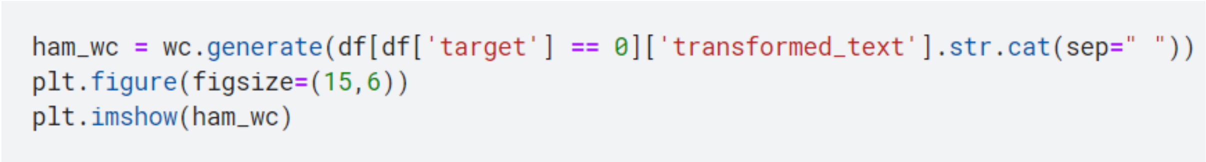
```
df['transformed_text'] = df['text'].apply(transform_text)
df.head()
```

	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	23	2	go jurong point avail bugi n great world la e ...
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 fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah think goe usf live around though

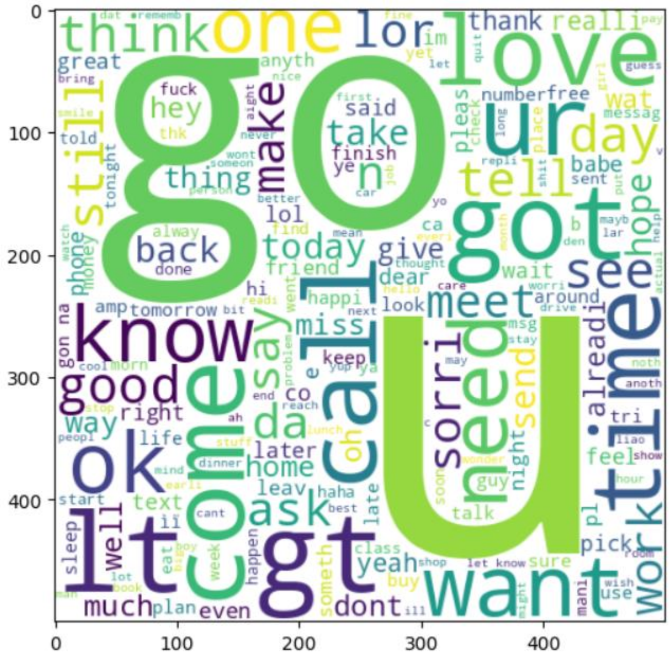


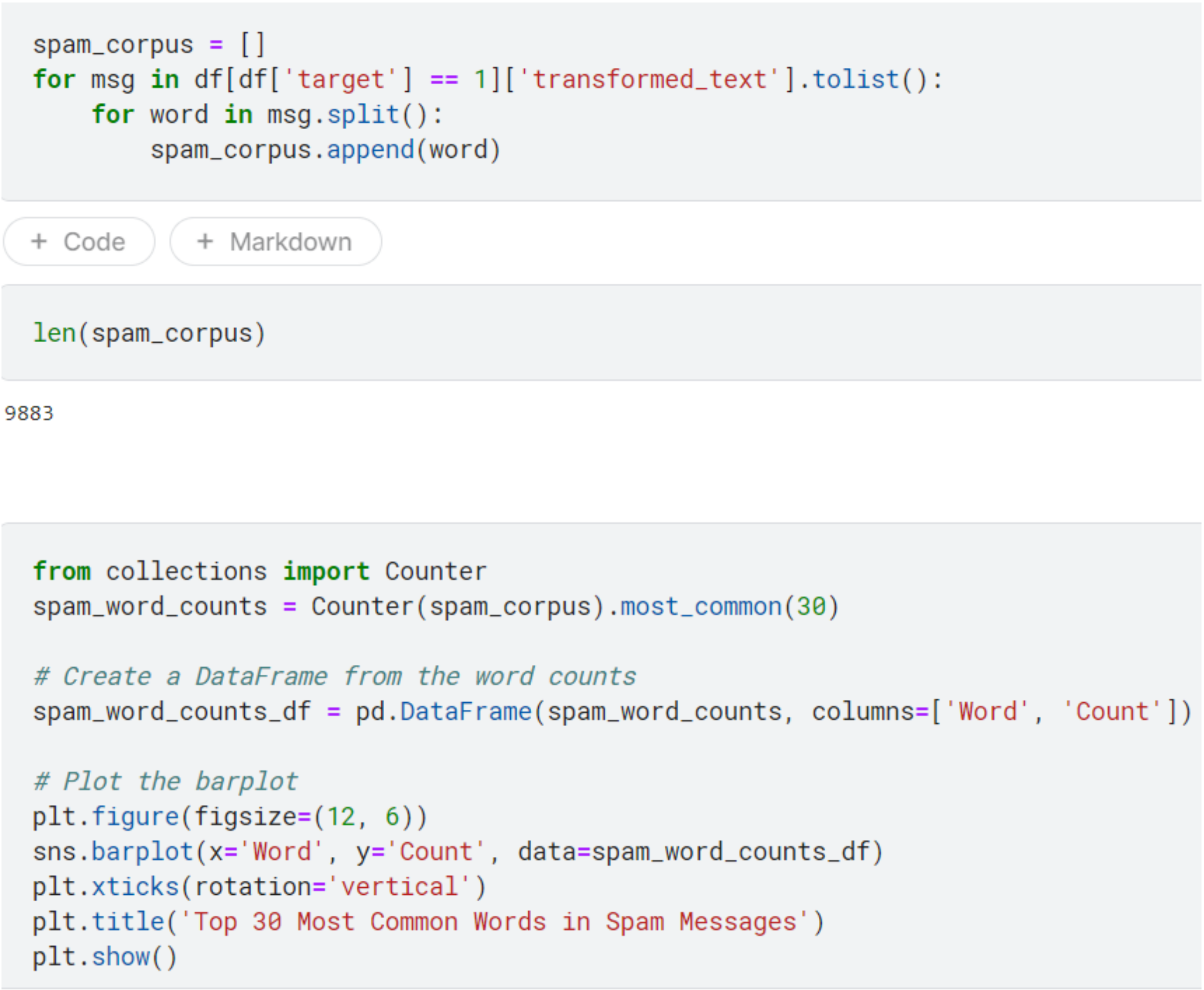
```
from wordcloud import WordCloud
wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
spam_wc = wc.generate(df[df['target'] == 1]['transformed_text'].str.cat(sep=" "))

plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
```





```
ham_wc = wc.generate(df[df['target'] == 0]['transformed_text'].str.cat(sep=" "))
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
```





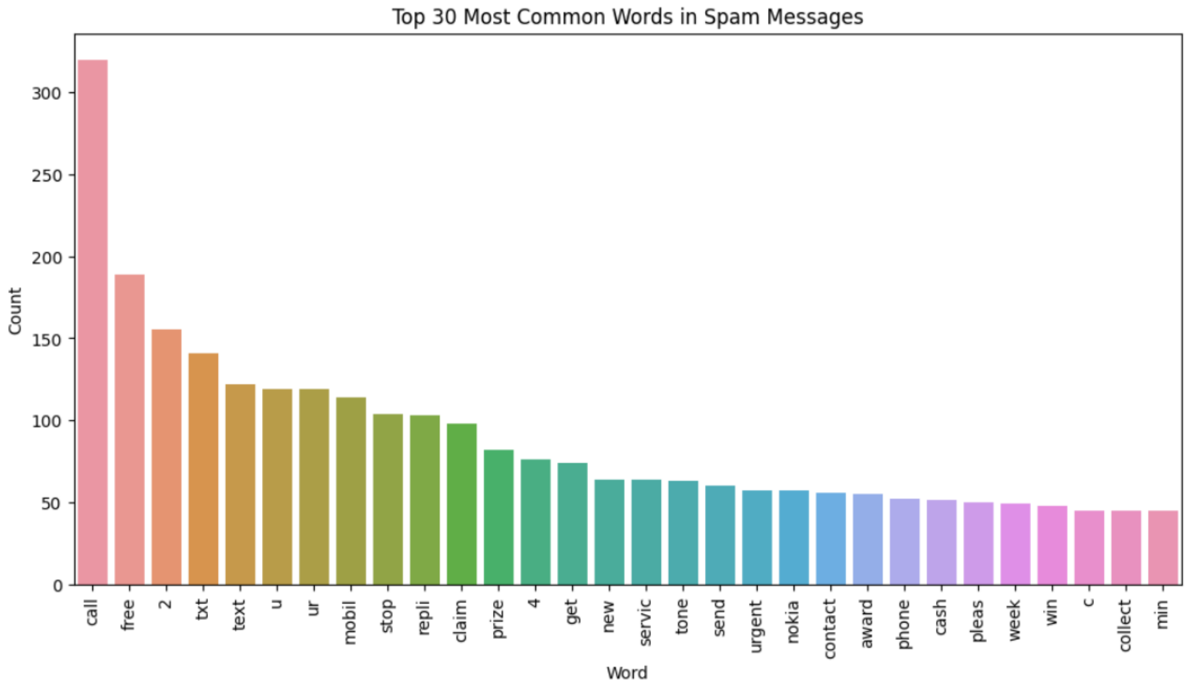
```
spam_corpus = []
for msg in df[df['target'] == 1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)
```

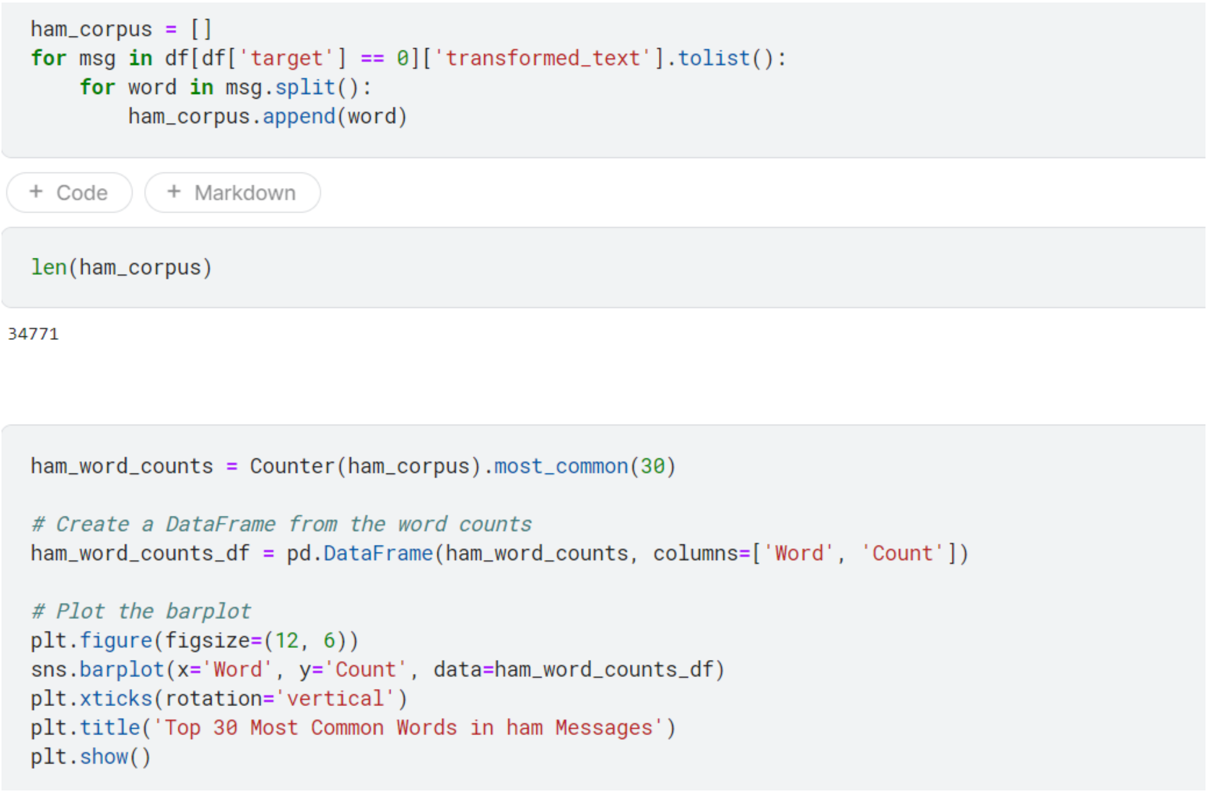
+ Code + Markdown

```
len(spam_corpus)
```

```
9883
```

```
from collections import Counter
spam_word_counts = Counter(spam_corpus).most_common(30)

# Create a DataFrame from the word counts
spam_word_counts_df = pd.DataFrame(spam_word_counts, columns=['Word', 'Count'])

# Plot the barplot
plt.figure(figsize=(12, 6))
sns.barplot(x='Word', y='Count', data=spam_word_counts_df)
plt.xticks(rotation='vertical')
plt.title('Top 30 Most Common Words in Spam Messages')
plt.show()
```



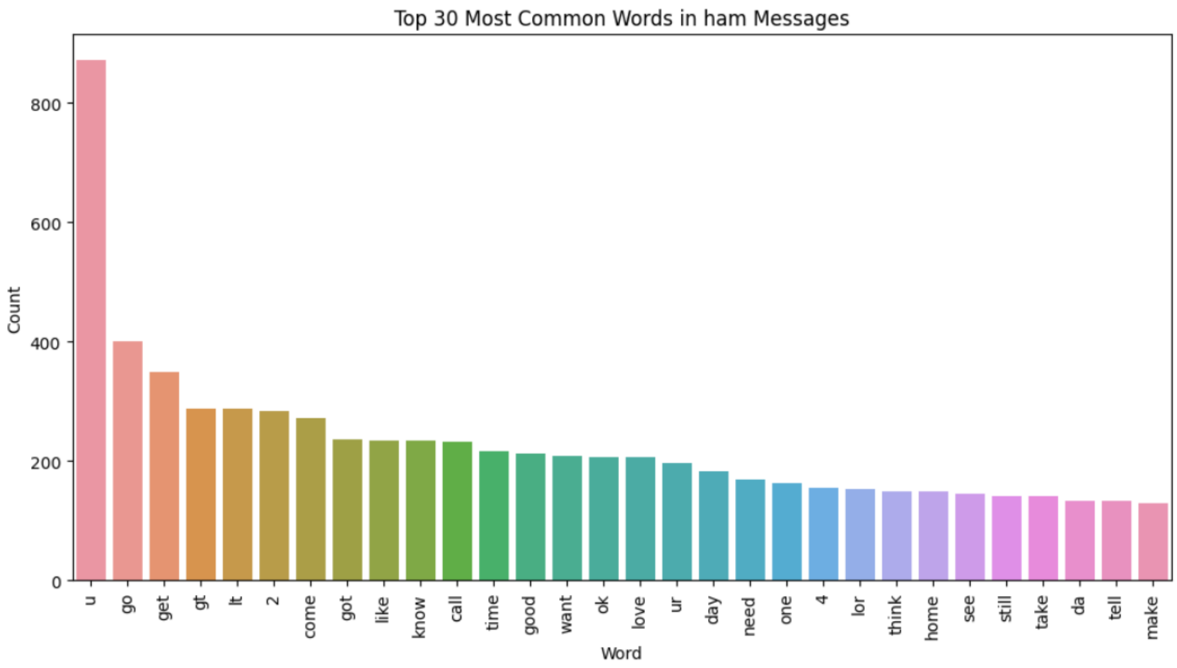


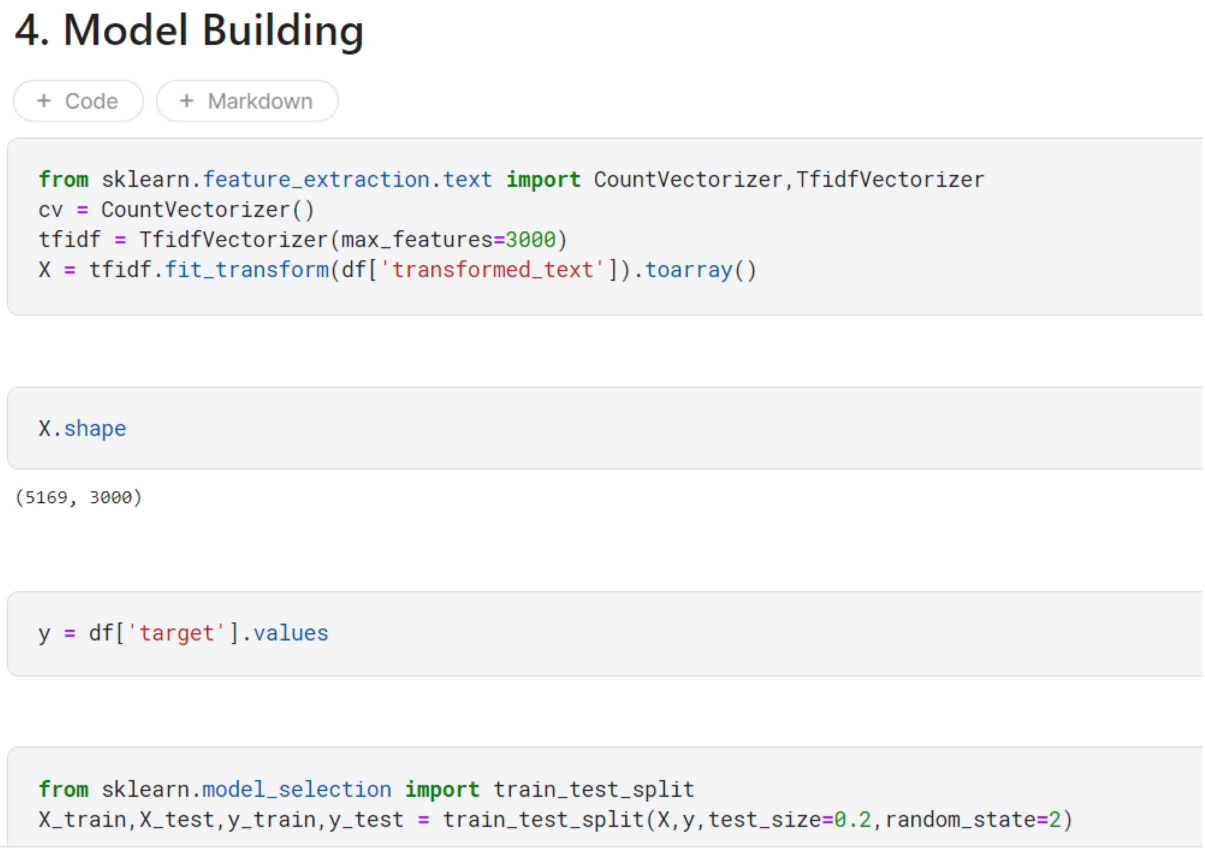
```
ham_corpus = []
for msg in df[df['target'] == 0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
```

```
len(ham_corpus)
```

```
34771
```

```
ham_word_counts = Counter(ham_corpus).most_common(30)

# Create a DataFrame from the word counts
ham_word_counts_df = pd.DataFrame(ham_word_counts, columns=['Word', 'Count'])

# Plot the barplot
plt.figure(figsize=(12, 6))
sns.barplot(x='Word', y='Count', data=ham_word_counts_df)
plt.xticks(rotation='vertical')
plt.title('Top 30 Most Common Words in ham Messages')
plt.show()
```





4. Model Building

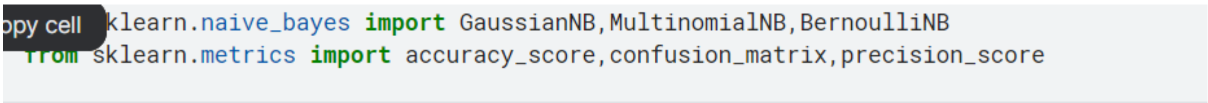
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
X = tfidf.fit_transform(df['transformed_text']).toarray()
```

```
X.shape
```

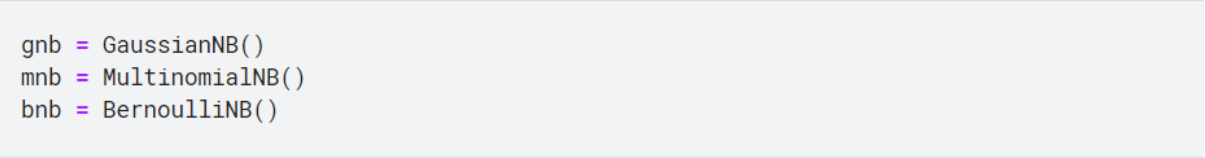
```
(5169, 3000)
```

```
y = df['target'].values
```

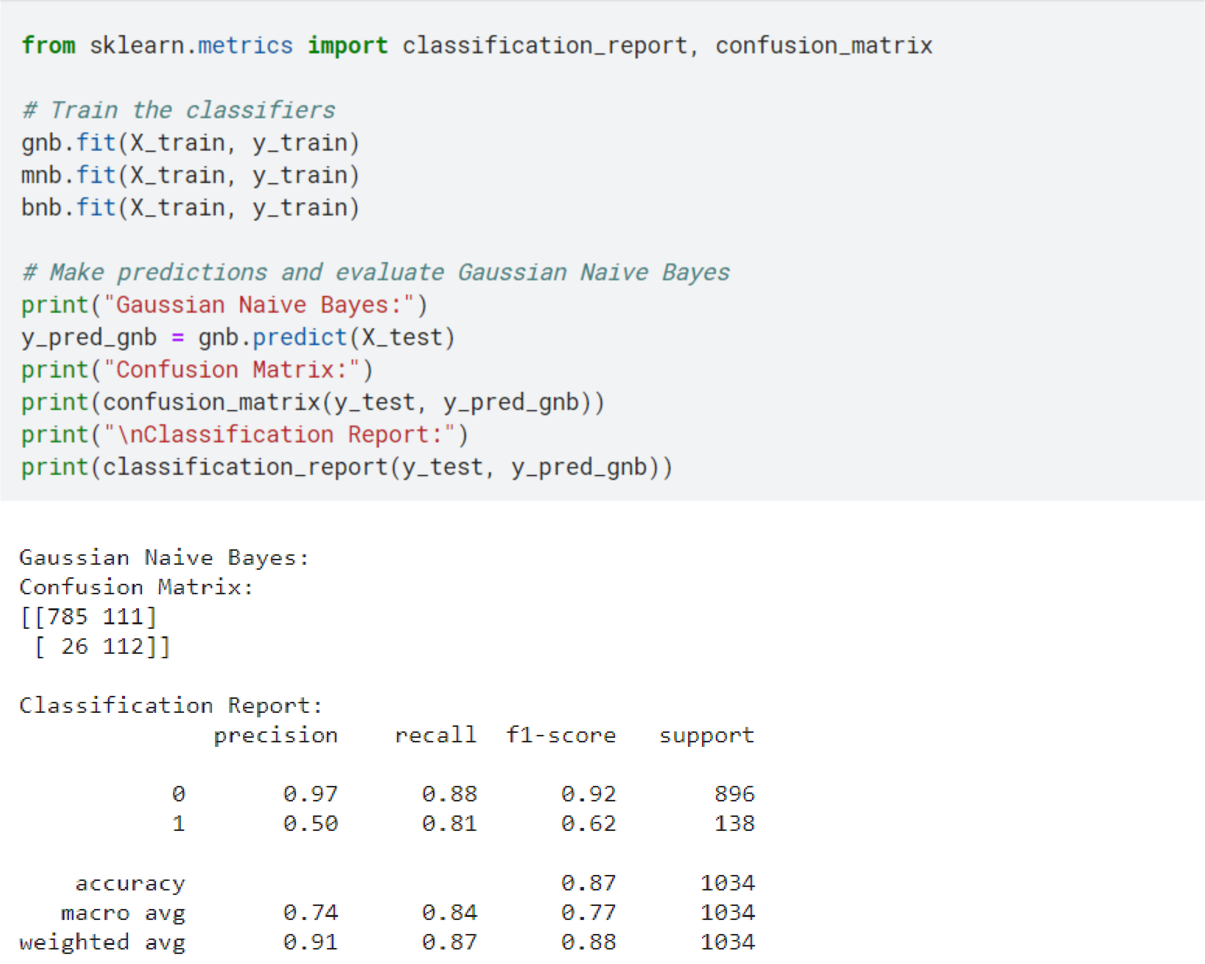
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```



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

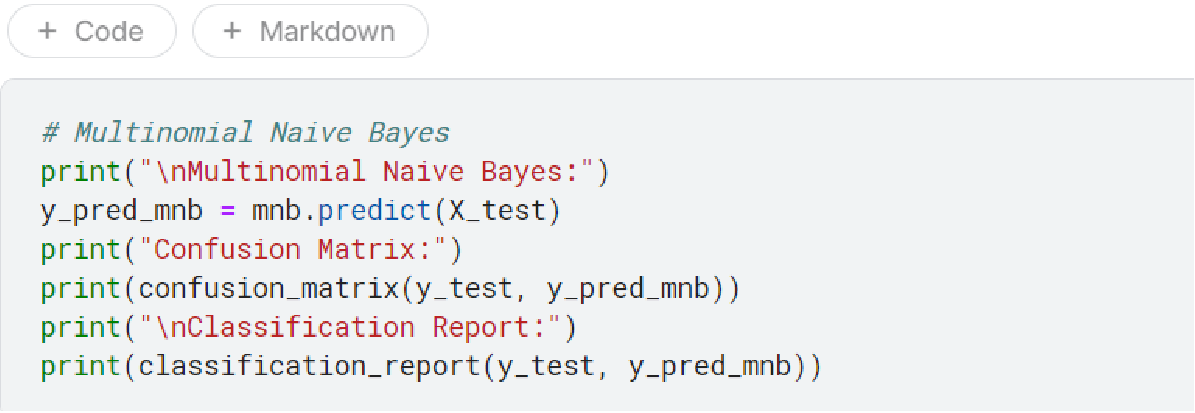


```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

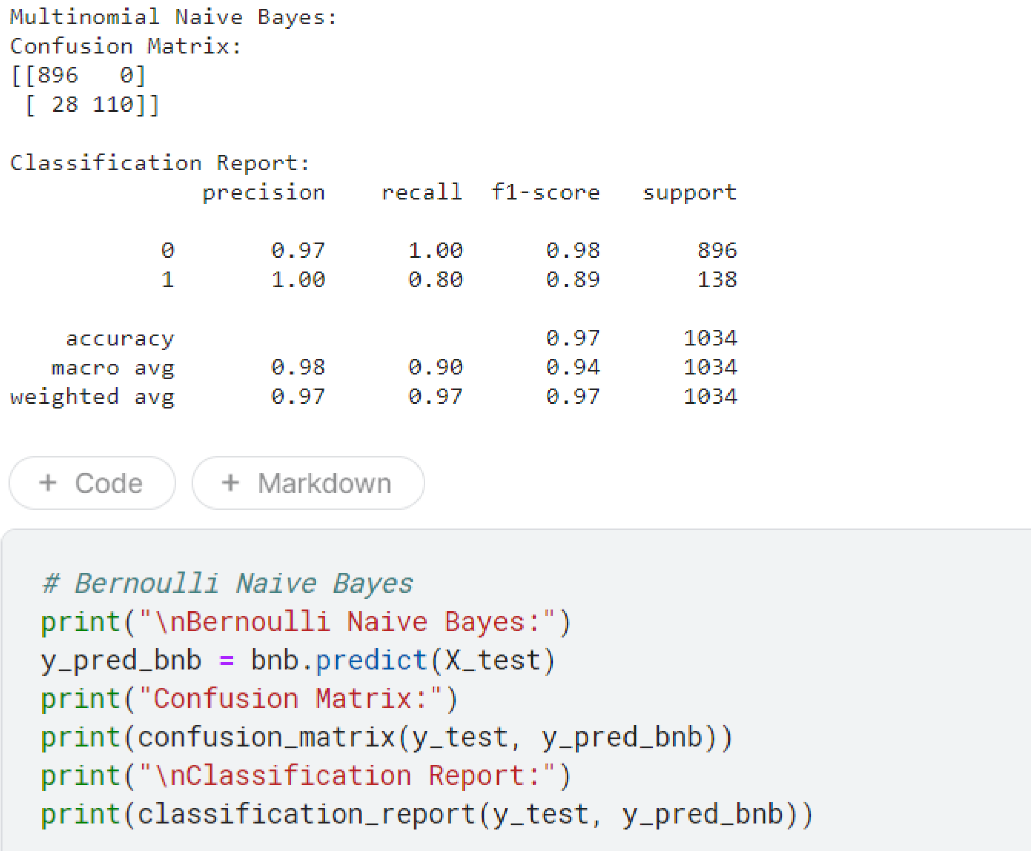


```
from sklearn.metrics import classification_report, confusion_matrix

# Train the classifiers
gnb.fit(X_train, y_train)
mnb.fit(X_train, y_train)
bnb.fit(X_train, y_train)

# Make predictions and evaluate Gaussian Naive Bayes
print("Gaussian Naive Bayes:")
y_pred_gnb = gnb.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_gnb))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_gnb))
```

```
Gaussian Naive Bayes:
Confusion Matrix:
[[785 111]
 [ 26 112]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.88      0.92       896
           1       0.50      0.81      0.62       138

    accuracy                           0.87      1034
   macro avg       0.74      0.84      0.77      1034
weighted avg       0.91      0.87      0.88      1034
```

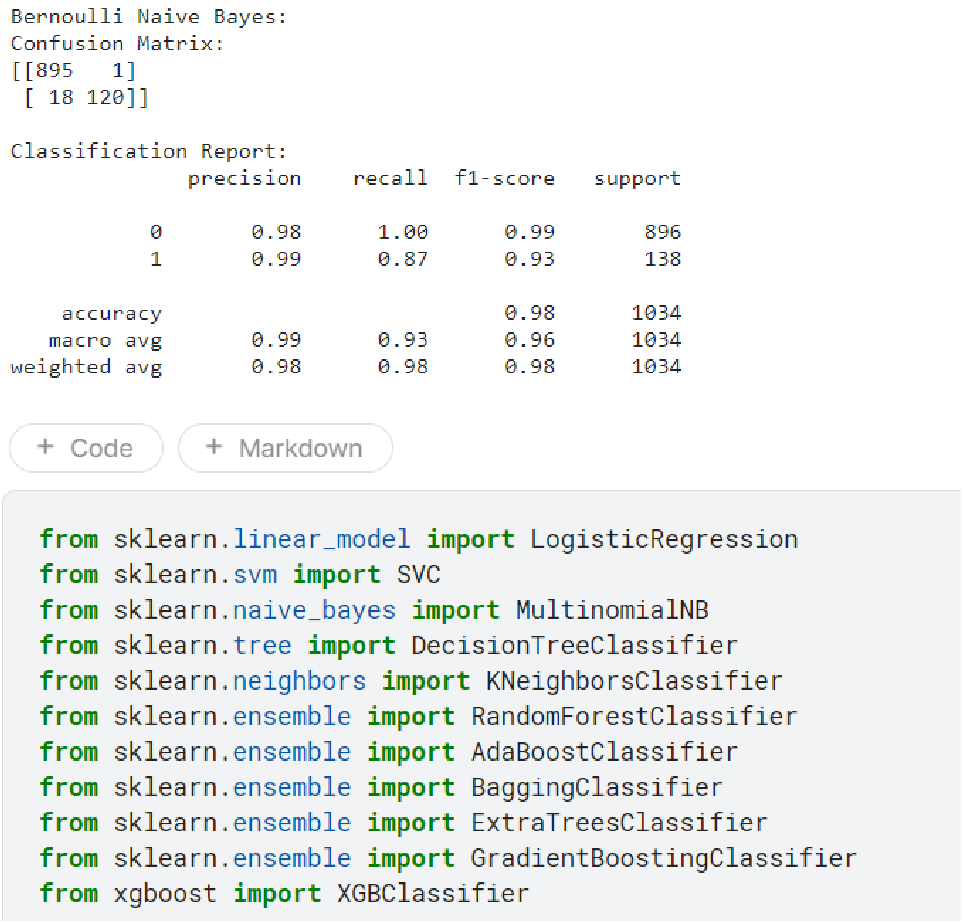


```
# Multinomial Naive Bayes
print("\nMultinomial Naive Bayes:")
y_pred_mnb = mnb.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_mnb))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_mnb))
```



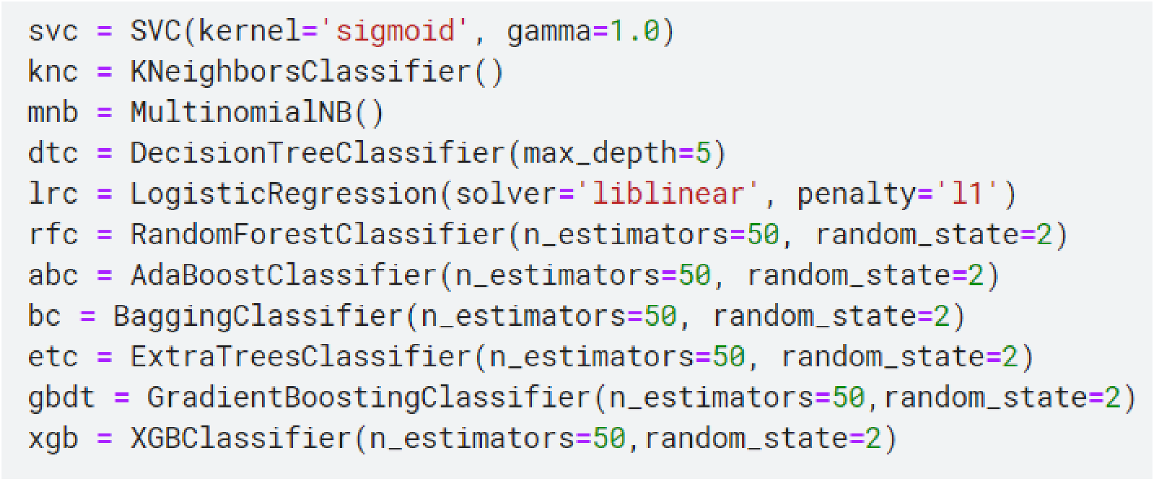
```
Multinomial Naive Bayes:
Confusion Matrix:
[[896   0]
 [ 28 110]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       896
           1       1.00      0.80      0.89       138

    accuracy                           0.97      1034
   macro avg       0.98      0.90      0.94      1034
weighted avg       0.97      0.97      0.97      1034
```

```python
# Bernoulli Naive Bayes
print("\nBernoulli Naive Bayes:")
y_pred_bnb = bnb.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_bnb))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_bnb))
```

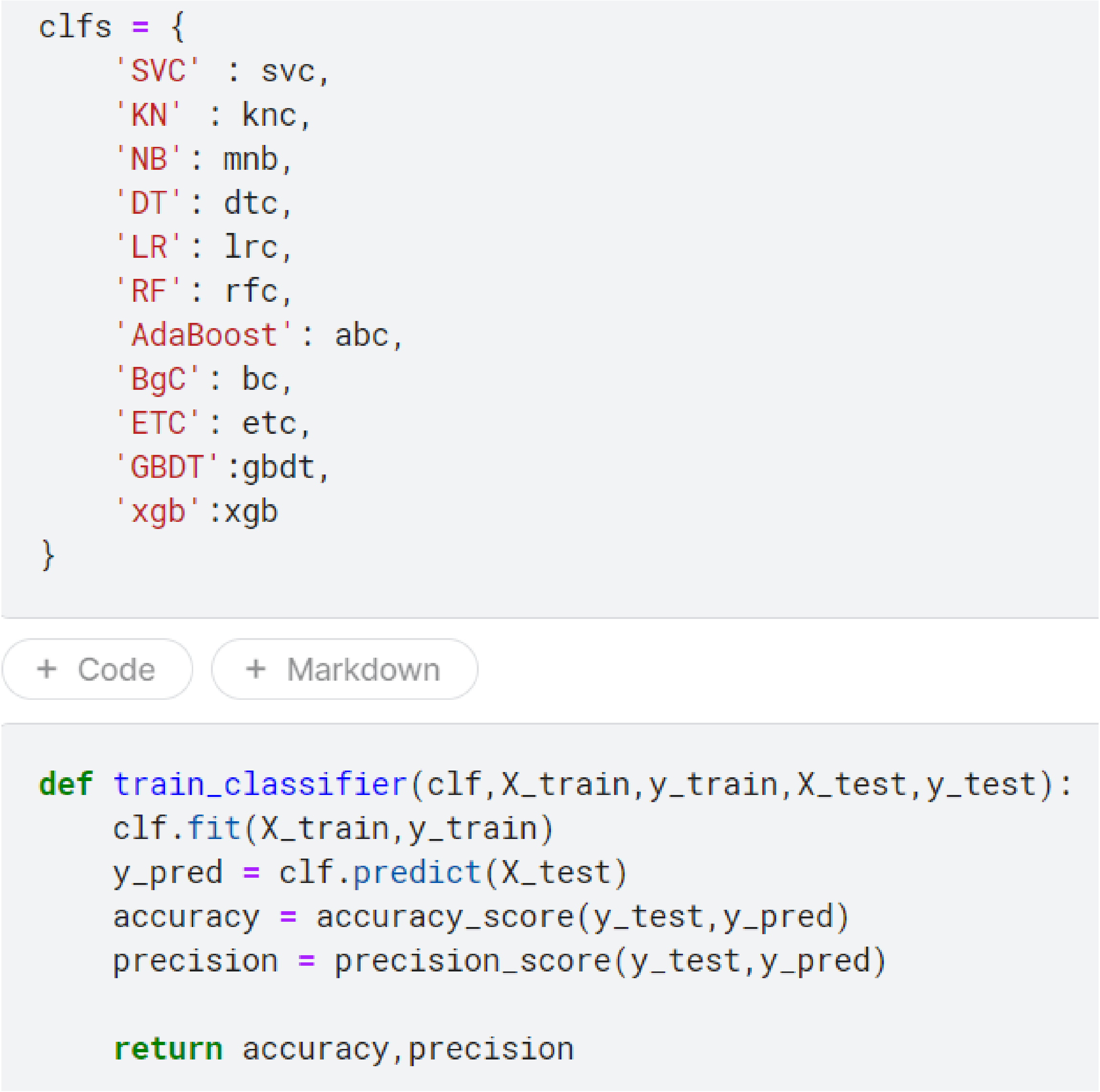


```
Bernoulli Naive Bayes:
Confusion Matrix:
[[895   1]
 [ 18 120]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       896
           1       0.99      0.87      0.93       138

    accuracy                           0.98      1034
   macro avg       0.99      0.93      0.96      1034
weighted avg       0.98      0.98      0.98      1034
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

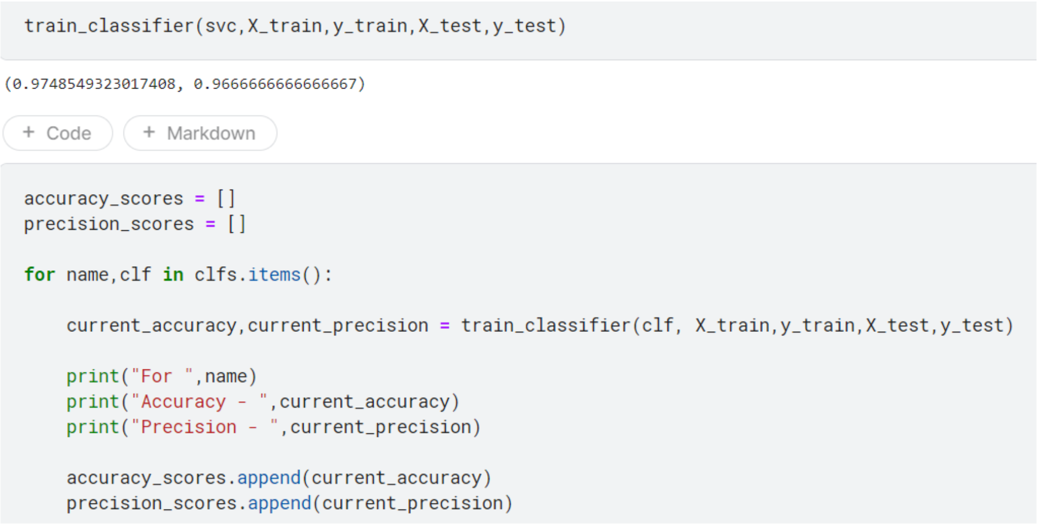


```
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier()
mnb = MultinomialNB()
dtc = DecisionTreeClassifier(max_depth=5)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
abc = AdaBoostClassifier(n_estimators=50, random_state=2)
bc = BaggingClassifier(n_estimators=50, random_state=2)
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
xgb = XGBClassifier(n_estimators=50,random_state=2)
```



```
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT':gbdt,
    'xgb':xgb
}
```

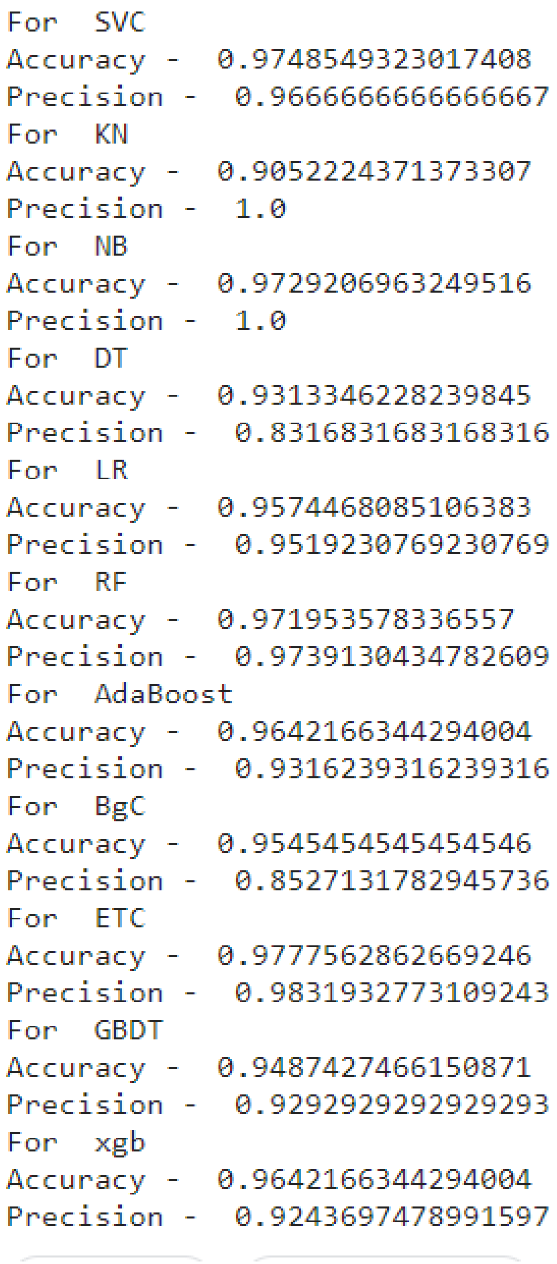
```
def train_classifier(clf,X_train,y_train,X_test,y_test):
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred)

    return accuracy,precision
```



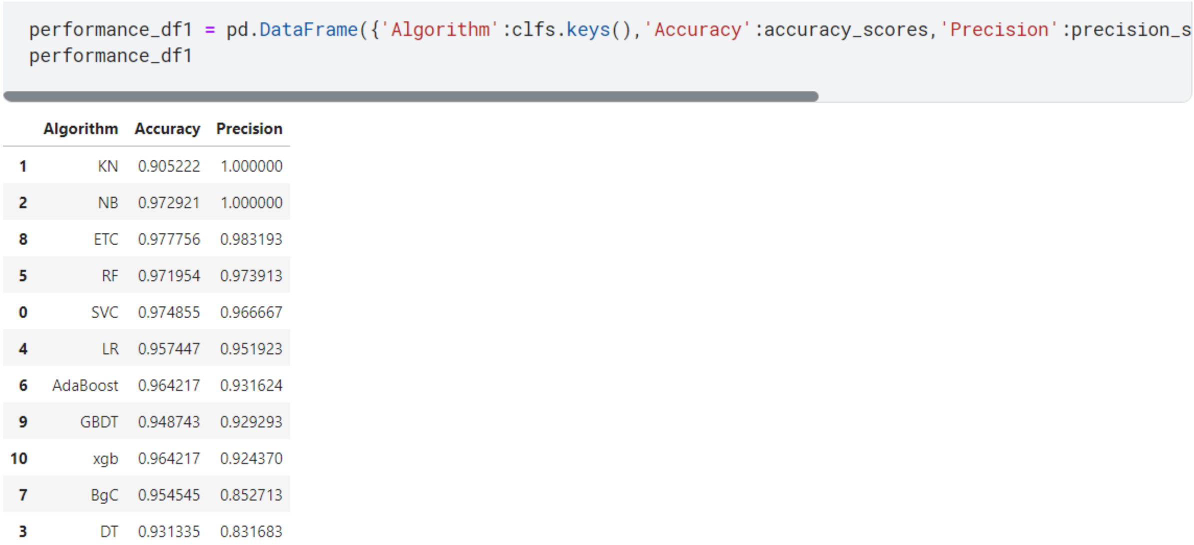
```
train_classifier(svc,X_train,y_train,X_test,y_test)
```

```
(0.9748549323017408, 0.9666666666666667)
```

```
accuracy_scores = []
precision_scores = []

for name,clf in clfs.items():

    current_accuracy,current_precision = train_classifier(clf, X_train,y_train,X_test,y_test)

    print("For ",name)
    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)

    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

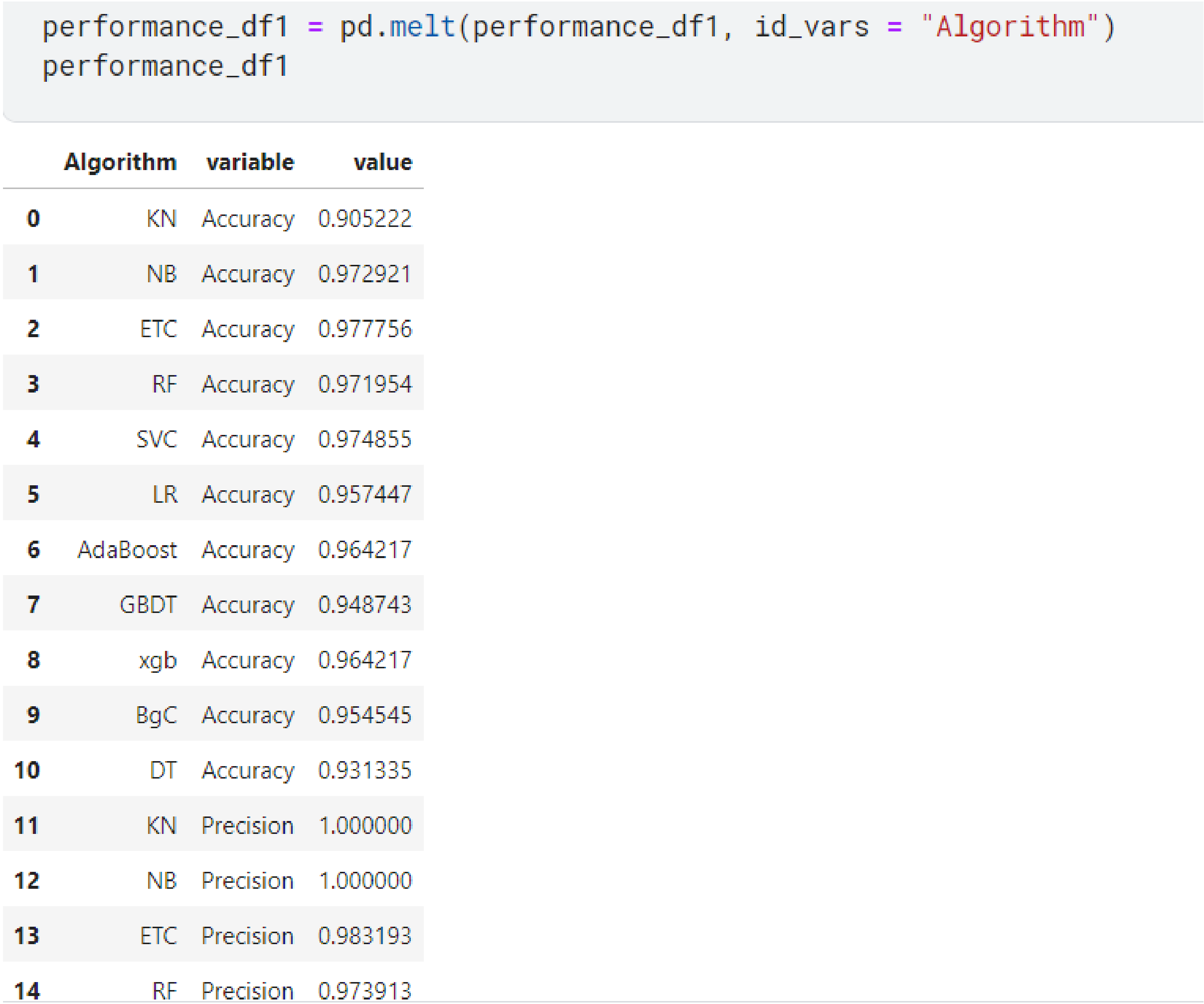


```
For  SVC
Accuracy -  0.9748549323017408
Precision -  0.9666666666666667
For  KN
Accuracy -  0.9052224371373307
Precision -  1.0
For  NB
Accuracy -  0.9729206963249516
Precision -  1.0
For  DT
Accuracy -  0.9313346228239845
Precision -  0.8316831683168316
For  LR
Accuracy -  0.9574468085106383
Precision -  0.9519230769230769
For  RF
Accuracy -  0.971953578336557
Precision -  0.9739130434782609
For  AdaBoost
Accuracy -  0.9642166344294004
Precision -  0.9316239316239316
For  BgC
Accuracy -  0.9545454545454546
Precision -  0.8527131782945736
For  ETC
Accuracy -  0.9777562862669246
Precision -  0.9831932773109243
For  GBDT
Accuracy -  0.9487427466150871
Precision -  0.9292929292929293
For  xgb
Accuracy -  0.9642166344294004
Precision -  0.9243697478991597
```



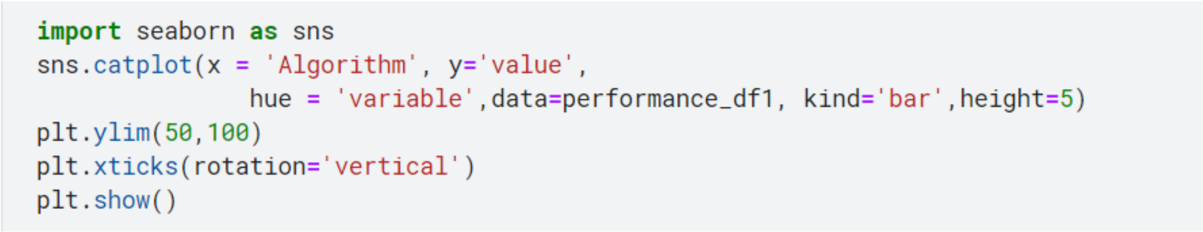
```
performance_df1 = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_scores,'Precision':precision_s
performance_df1
```

	Algorithm	Accuracy	Precision
1	KN	0.905222	1.000000
2	NB	0.972921	1.000000
8	ETC	0.977756	0.983193
5	RF	0.971954	0.973913
0	SVC	0.974855	0.966667
4	LR	0.957447	0.951923
6	AdaBoost	0.964217	0.931624
9	GBDT	0.948743	0.929293
10	xgb	0.964217	0.924370
7	BgC	0.954545	0.852713
3	DT	0.931335	0.831683

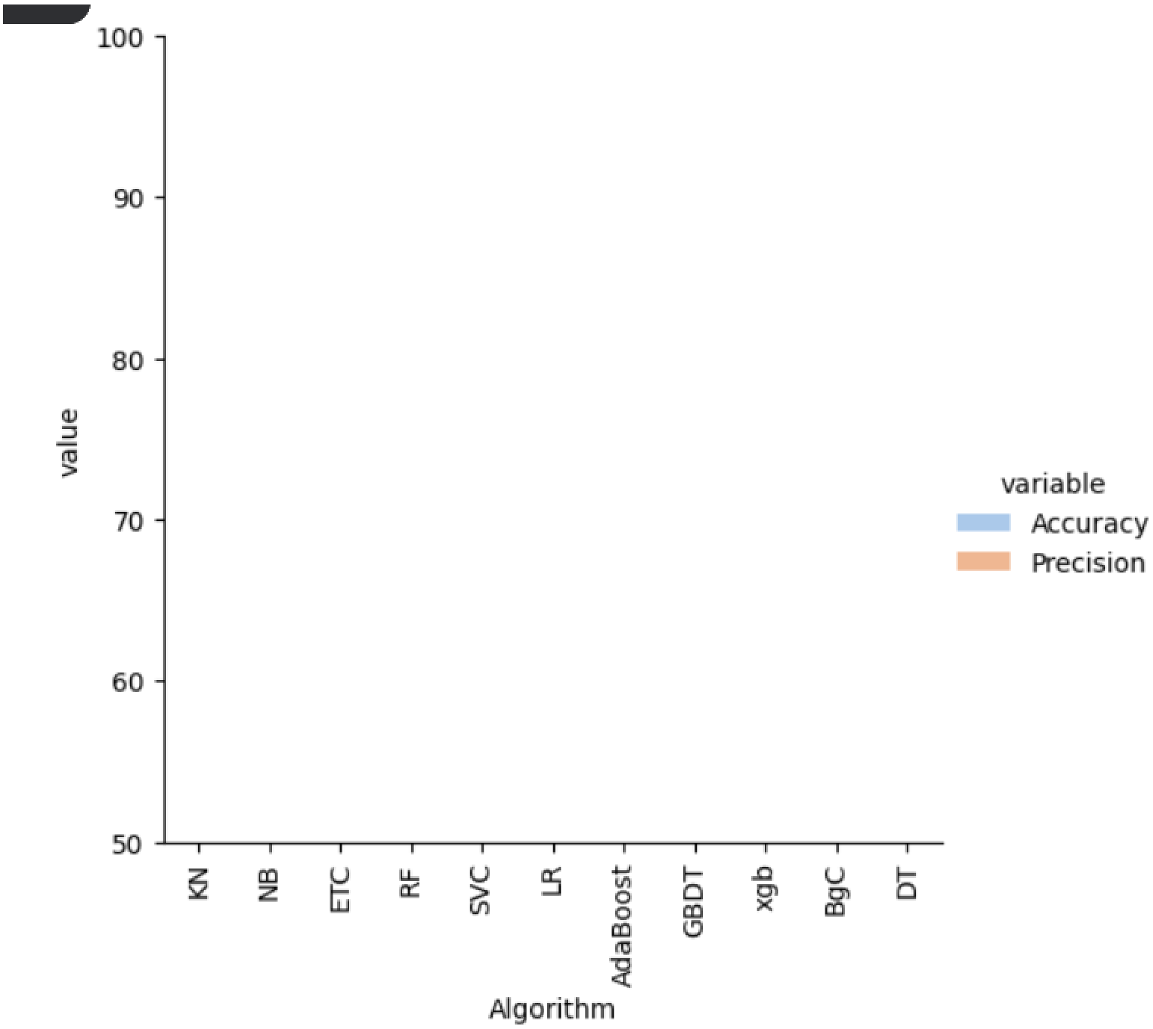


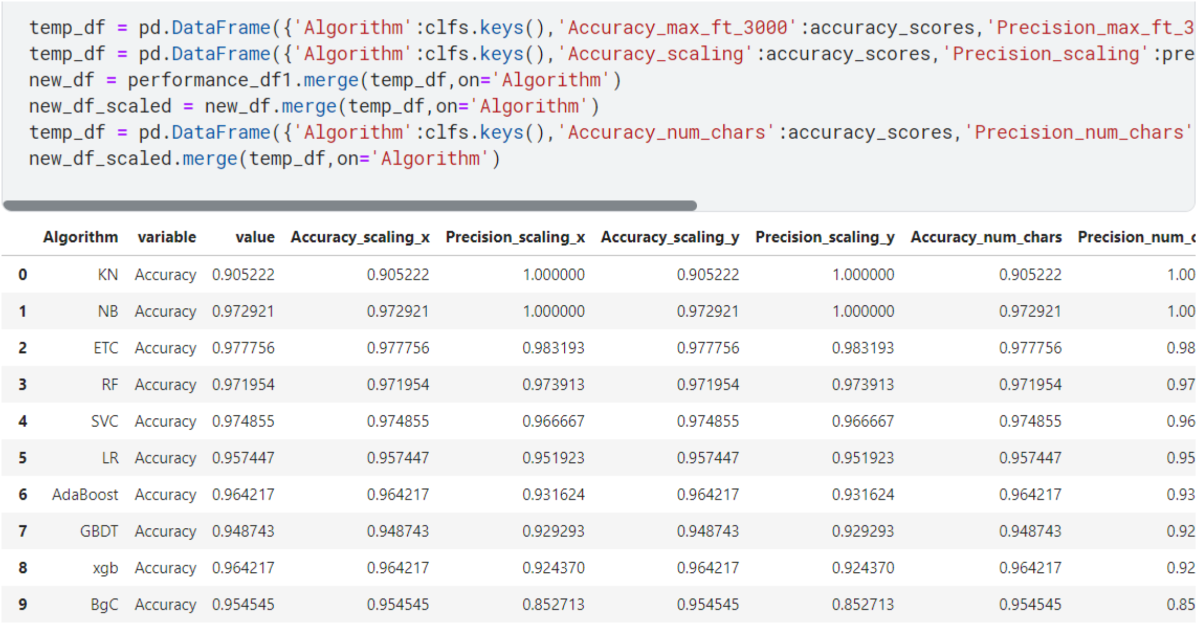
```
performance_df1 = pd.melt(performance_df1, id_vars = "Algorithm")
performance_df1
```

	Algorithm	variable	value
0	KN	Accuracy	0.905222
1	NB	Accuracy	0.972921
2	ETC	Accuracy	0.977756
3	RF	Accuracy	0.971954
4	SVC	Accuracy	0.974855
5	LR	Accuracy	0.957447
6	AdaBoost	Accuracy	0.964217
7	GBDT	Accuracy	0.948743
8	xgb	Accuracy	0.964217
9	BgC	Accuracy	0.954545
10	DT	Accuracy	0.931335
11	KN	Precision	1.000000
12	NB	Precision	1.000000
13	ETC	Precision	0.983193
14	RF	Precision	0.973913



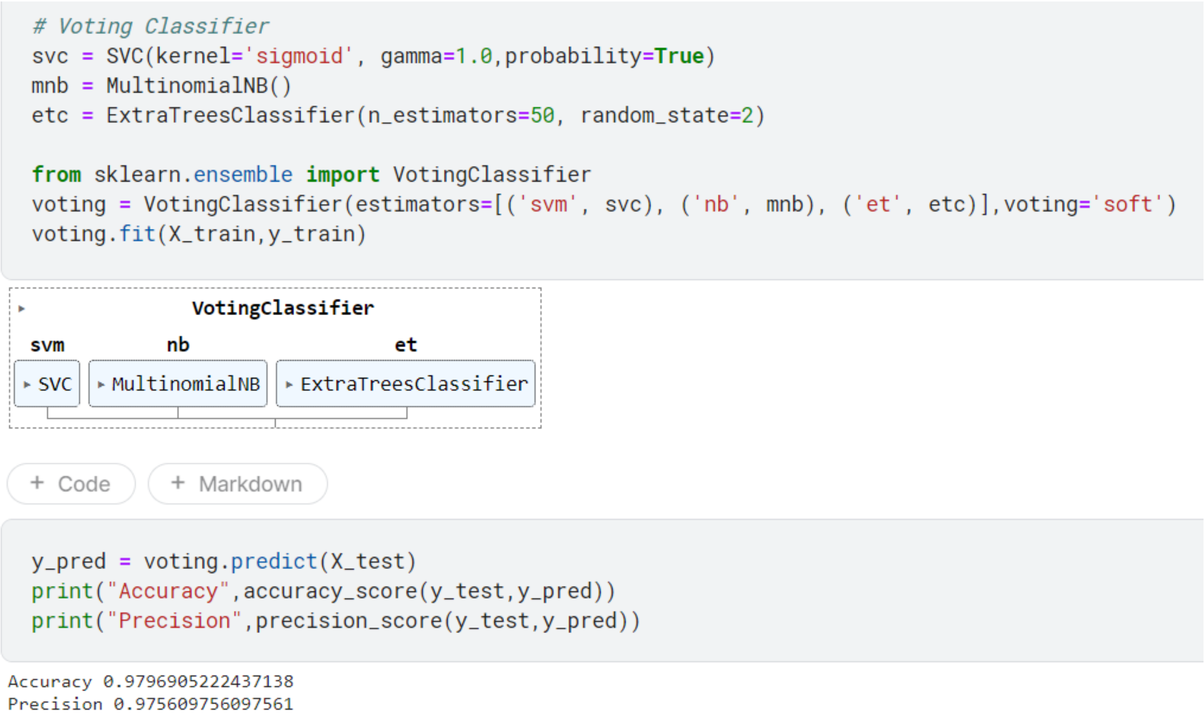
```
import seaborn as sns
sns.catplot(x = 'Algorithm', y='value',
               hue = 'variable',data=performance_df1, kind='bar',height=5)
plt.ylim(50,100)
plt.xticks(rotation='vertical')
plt.show()
```





```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':accuracy_scores,'Precision_max_ft_3
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_scaling':accuracy_scores,'Precision_scaling':pre
new_df = performance_df1.merge(temp_df,on='Algorithm')
new_df_scaled = new_df.merge(temp_df,on='Algorithm')
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':accuracy_scores,'Precision_num_chars'
new_df_scaled.merge(temp_df,on='Algorithm')
```

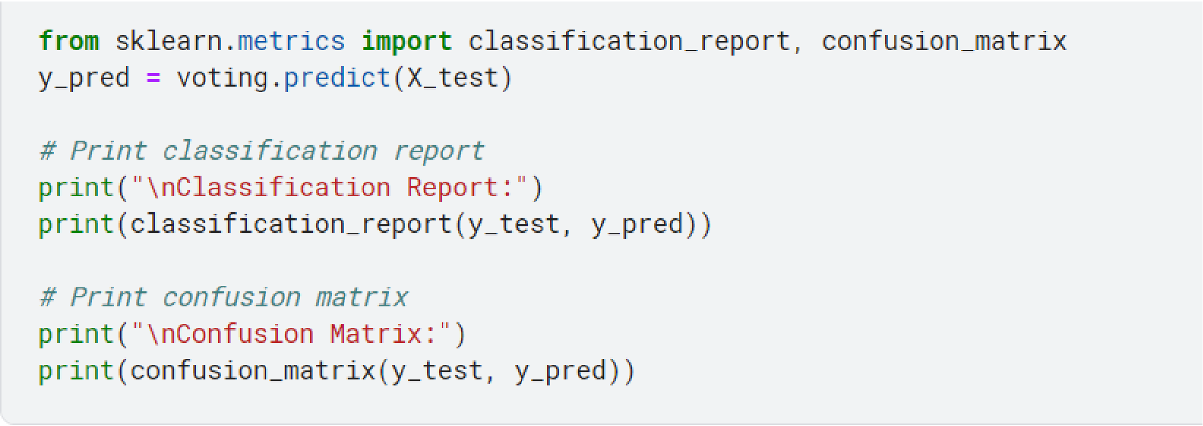
	Algorithm	variable	value	Accuracy_scaling_x	Precision_scaling_x	Accuracy_scaling_y	Precision_scaling_y	Accuracy_num_chars	Precision_num_c
0	KN	Accuracy	0.905222	0.905222	1.000000	0.905222	1.000000	0.905222	1.00
1	NB	Accuracy	0.972921	0.972921	1.000000	0.972921	1.000000	0.972921	1.00
2	ETC	Accuracy	0.977756	0.977756	0.983193	0.977756	0.983193	0.977756	0.98
3	RF	Accuracy	0.971954	0.971954	0.973913	0.971954	0.973913	0.971954	0.97
4	SVC	Accuracy	0.974855	0.974855	0.966667	0.974855	0.966667	0.974855	0.96
5	LR	Accuracy	0.957447	0.957447	0.951923	0.957447	0.951923	0.957447	0.95
6	AdaBoost	Accuracy	0.964217	0.964217	0.931624	0.964217	0.931624	0.964217	0.93
7	GBDT	Accuracy	0.948743	0.948743	0.929293	0.948743	0.929293	0.948743	0.92
8	xgb	Accuracy	0.964217	0.964217	0.924370	0.964217	0.924370	0.964217	0.92
9	BgC	Accuracy	0.954545	0.954545	0.852713	0.954545	0.852713	0.954545	0.85



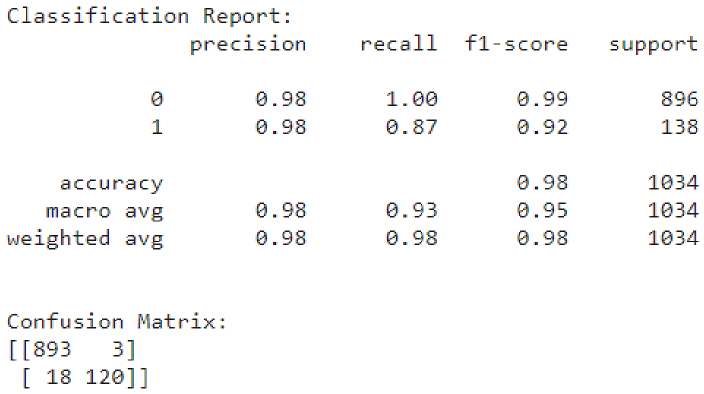
```
# Voting Classifier
svc = SVC(kernel='sigmoid', gamma=1.0,probability=True)
mnb = MultinomialNB()
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)

from sklearn.ensemble import VotingClassifier
voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et', etc)],voting='soft')
voting.fit(X_train,y_train)
```

```
y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

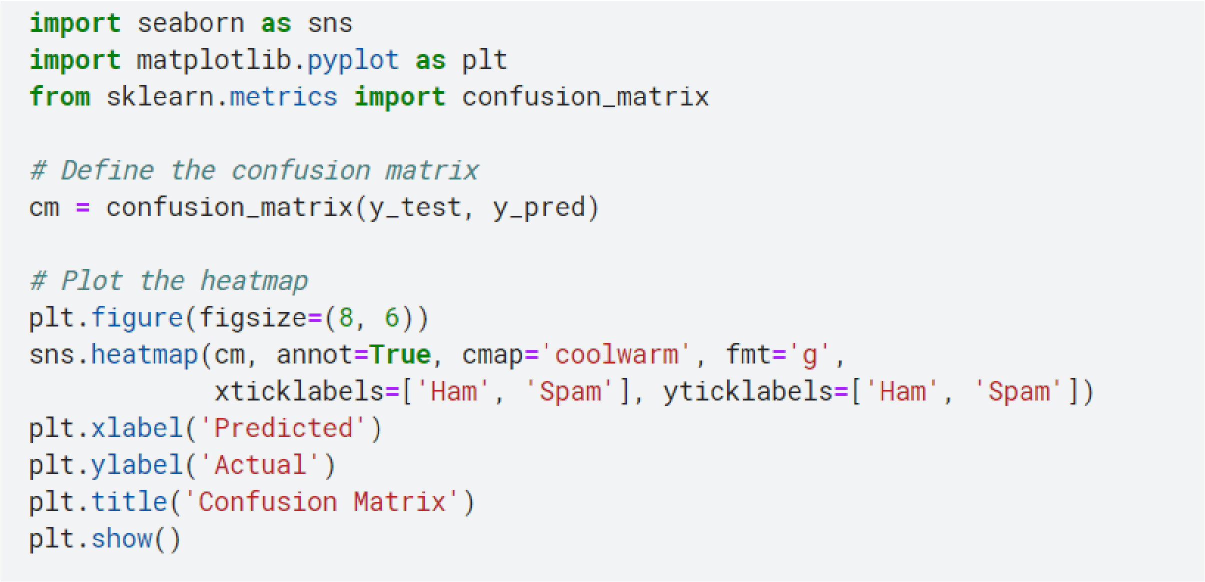
```
Accuracy 0.9796905222437138
Precision 0.975609756097561
```



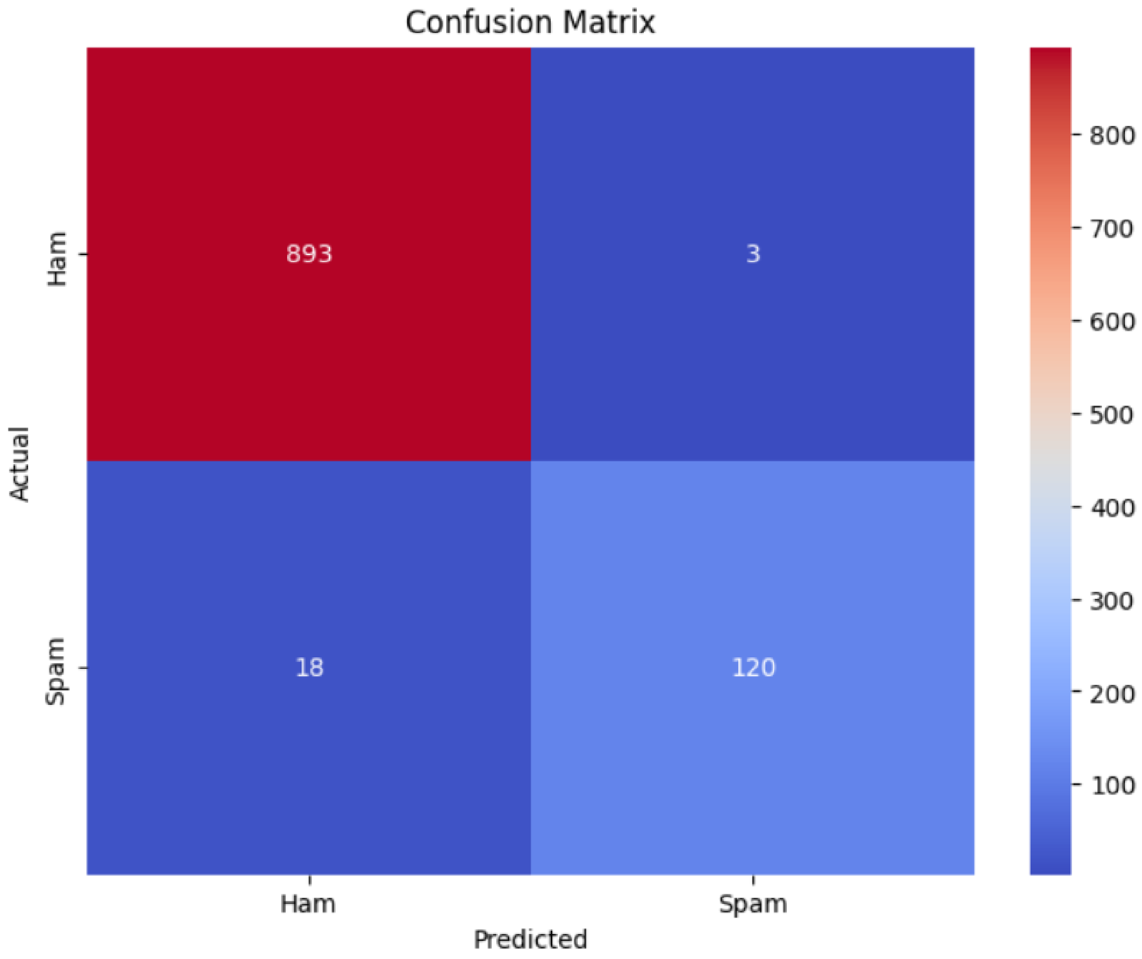
```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = voting.predict(X_test)

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

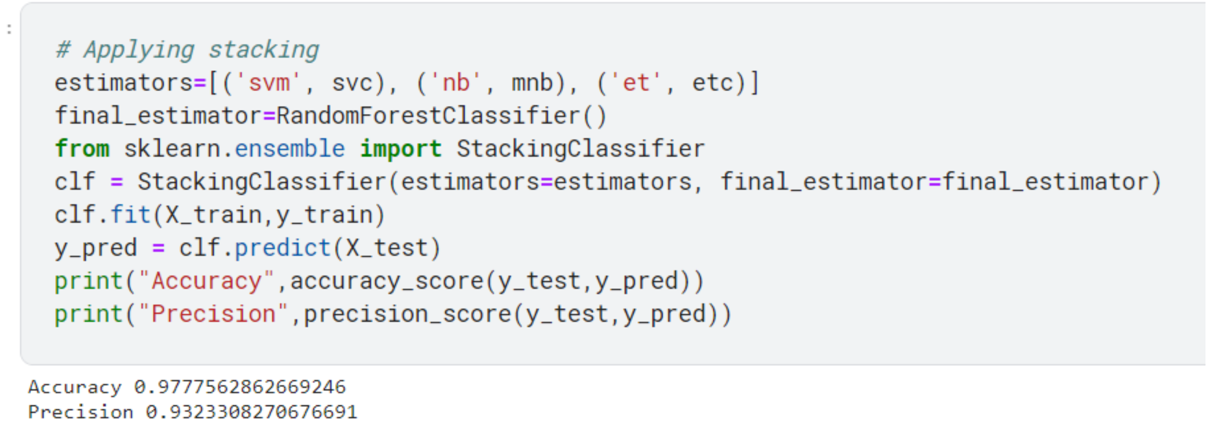


```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       896
           1       0.98      0.87      0.92       138

    accuracy                           0.98      1034
   macro avg       0.98      0.93      0.95      1034
weighted avg       0.98      0.98      0.98      1034


Confusion Matrix:
[[893   3]
 [ 18 120]]
```



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Define the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='coolwarm', fmt='g',
            xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```





```
# Applying stacking
estimators=[('svm', svc), ('nb', mnb), ('et', etc)]
final_estimator=RandomForestClassifier()
from sklearn.ensemble import StackingClassifier
clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

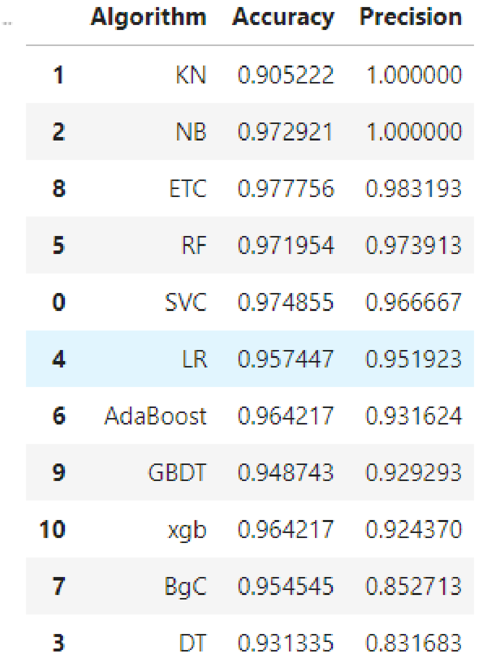
```
Accuracy 0.9777562862669246
Precision 0.9323308270676691
```

4.2. Model Performance Evaluation

When assessing the effectiveness of your email spam detector, it's important to consider several key metrics that provide insight into its performance. One crucial metric is accuracy, which reveals how often the detector makes correct classifications. Essentially, accuracy measures the proportion of correctly identified emails—whether they're spam or legitimate—out of all the emails it examines. This metric gives you a general sense of how well the detector is able to differentiate between spam and regular emails.

Precision is another significant metric in the evaluation process. Precision focuses on minimizing false alarms by examining how often the detector incorrectly labels a legitimate email as spam. It calculates the ratio of correctly identified spam emails to all the emails that the detector flagged as spam. High precision indicates that the detector is adept at avoiding unnecessary alerts, reducing the likelihood of legitimate emails being mistakenly classified as spam.

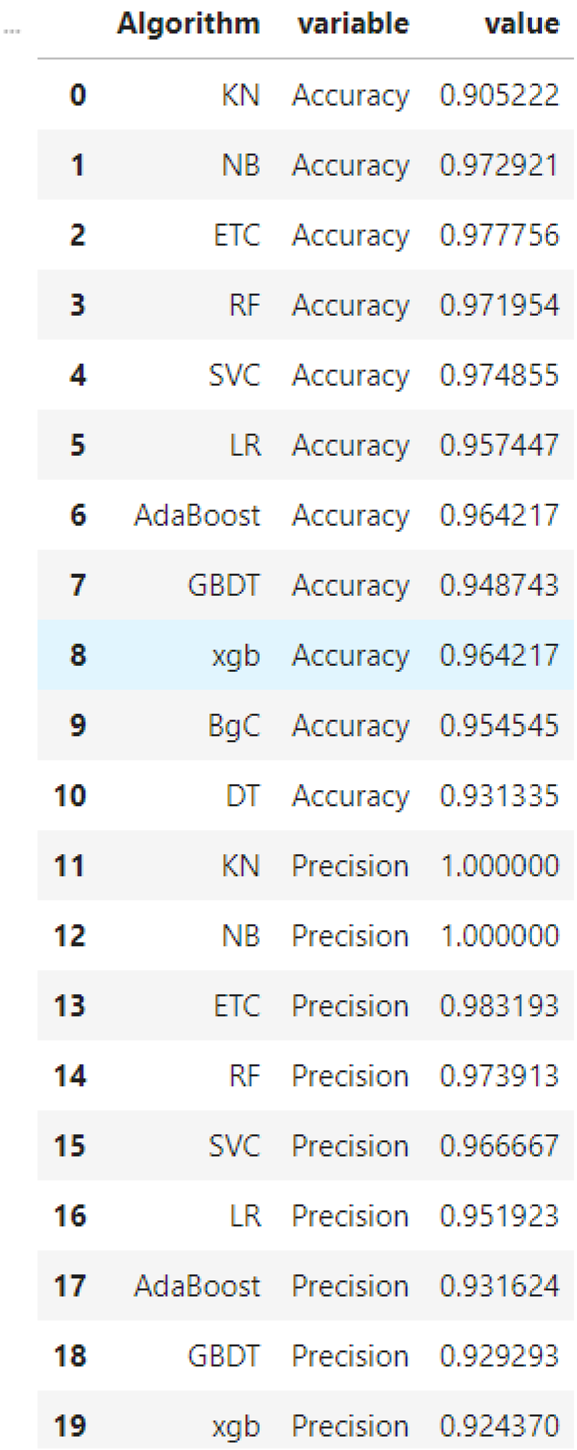
Moving on, recall, also referred to as sensitivity, assesses the detector's ability to capture all instances of real spam emails. It looks at the ratio of correctly identified spam emails to the total number of actual spam emails present in the dataset. A high recall score indicates that the detector is effective at identifying the majority of spam emails without overlooking any. This is crucial for ensuring that no potential threats slip through undetected.



	Algorithm	Accuracy	Precision
1	KN	0.905222	1.000000
2	NB	0.972921	1.000000
8	ETC	0.977756	0.983193
5	RF	0.971954	0.973913
0	SVC	0.974855	0.966667
4	LR	0.957447	0.951923
6	AdaBoost	0.964217	0.931624
9	GBDT	0.948743	0.929293
10	xgb	0.964217	0.924370
7	BgC	0.954545	0.852713
3	DT	0.931335	0.831683

Lastly, the F1-Score provides a balanced measure by combining precision and recall into a single metric. It offers a comprehensive evaluation of the detector's overall performance by considering both aspects simultaneously. The F1-Score is particularly useful when you want to assess the trade-off between precision and recall and determine the detector's effectiveness in striking a balance between them.

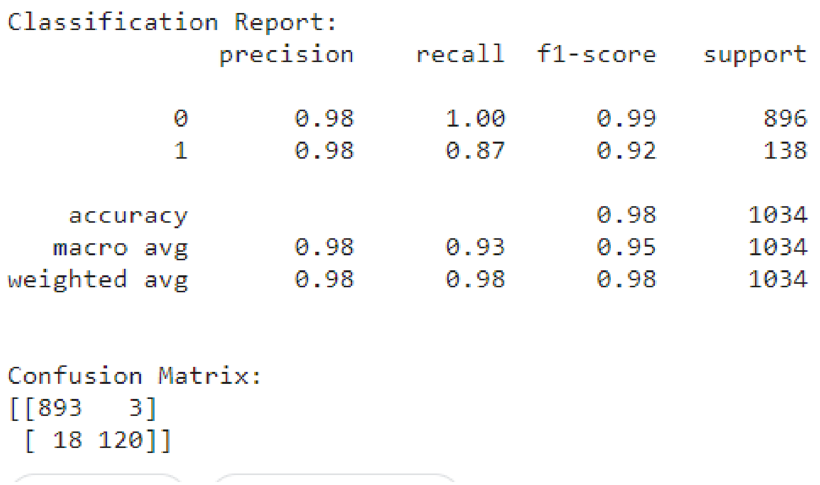
By analyzing these metrics, you gain valuable insights into how well your email spam detector functions. They help you understand its strengths and weaknesses, enabling you to make informed decisions about optimizing its performance and enhancing its ability to accurately identify and classify spam emails while minimizing false alarms.



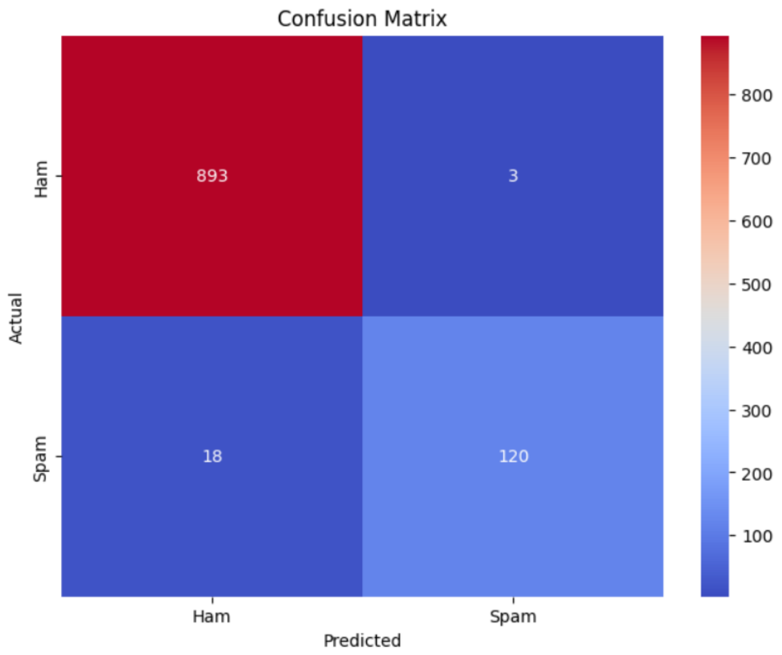
	Algorithm	variable	value
0	KN	Accuracy	0.905222
1	NB	Accuracy	0.972921
2	ETC	Accuracy	0.977756
3	RF	Accuracy	0.971954
4	SVC	Accuracy	0.974855
5	LR	Accuracy	0.957447
6	AdaBoost	Accuracy	0.964217
7	GBDT	Accuracy	0.948743
8	xgb	Accuracy	0.964217
9	BgC	Accuracy	0.954545
10	DT	Accuracy	0.931335
11	KN	Precision	1.000000
12	NB	Precision	1.000000
13	ETC	Precision	0.983193
14	RF	Precision	0.973913
15	SVC	Precision	0.966667
16	LR	Precision	0.951923
17	AdaBoost	Precision	0.931624
18	GBDT	Precision	0.929293
19	xgb	Precision	0.924370

4.2.1. Evaluation Metrics

In the case of evaluating an email spam classifier using NLP, assessing its performance is crucial. Evaluation metrics help quantify how well the model can spot and predict spam emails. These metrics include accuracy, which tells us how often the model correctly identifies spam emails among all predictions. Precision measures how well the model avoids wrongly flagging legitimate emails as spam. It looks at the ratio of correctly identified spam emails to all emails flagged as spam, aiming for fewer false alarms. Recall, also known as sensitivity, checks if the model finds all the real spam emails. It looks at the ratio of correctly identified spam emails to all actual spam emails, ensuring it doesn't miss any.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       896
           1       0.98      0.87      0.92       138

    accuracy                           0.98      1034
   macro avg       0.98      0.93      0.95      1034
weighted avg       0.98      0.98      0.98      1034


Confusion Matrix:
[[893   3]
 [ 18 120]]
```



Lastly, the F1-Score combines precision and recall to give a balanced view of the model's overall performance. It's handy when you want to see how well the model does at both precision and recall at the same time. These metrics help gauge how effective the email spam classifier is at keeping your inbox clean from unwanted messages.

4.2.2. Deployment and Testing

In the deployment phase, the trained predictive models are integrated into the organization's email infrastructure or systems to enable real-time or batch predictions of incoming emails. The deployment process involves the following steps:

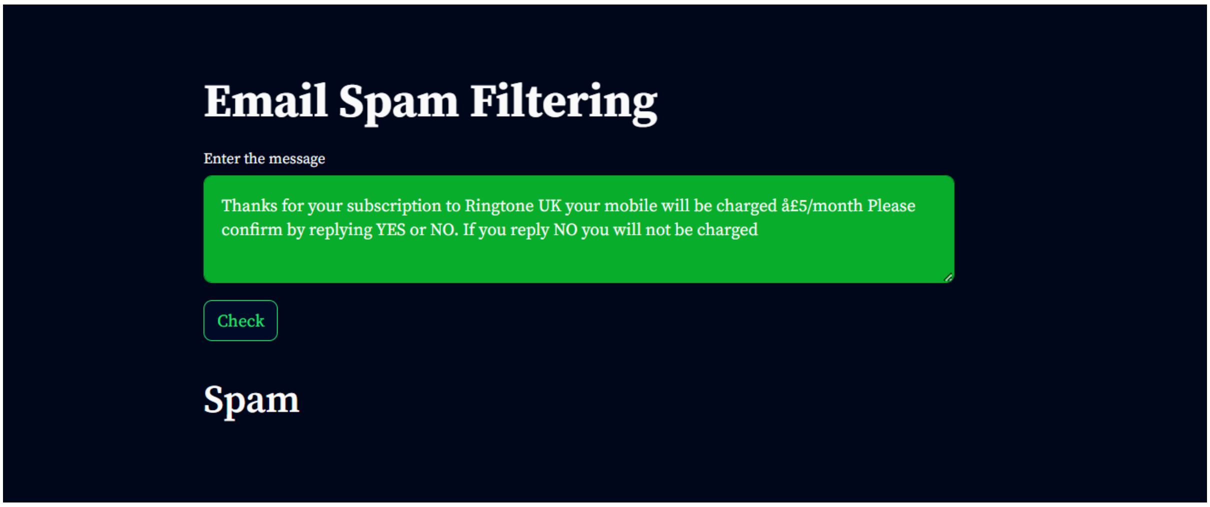
Integration with Email Servers: The predictive models are integrated with the organization's email servers or filtering systems to automatically classify incoming emails as spam or legitimate. This integration ensures seamless integration of the classifier into the email workflow.

API Development: APIs (Application Programming Interfaces) are developed to facilitate communication between the email servers and the predictive models. These APIs allow emails to be processed and classified in real-time, providing quick feedback to users.

Scalability and Performance Optimization: Measures are taken to ensure that the deployed models can handle the volume of incoming emails efficiently. Techniques such as load balancing, caching, and parallel processing are employed to optimize performance and scalability.

Testing in Real-world Scenarios:

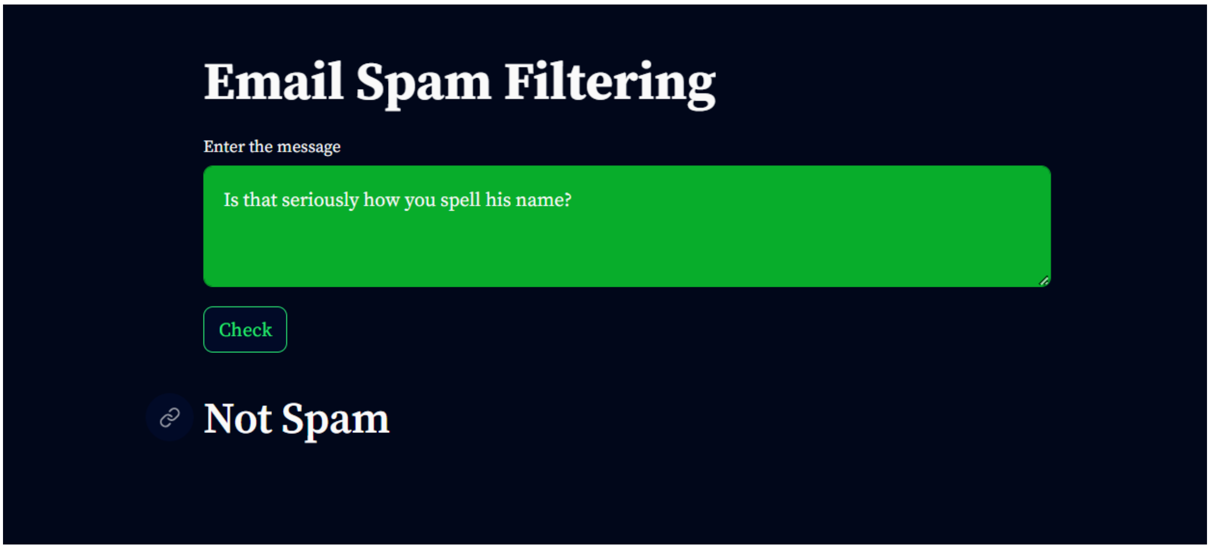
Real-world testing involves applying the deployed predictive models to live email data and evaluating their performance in identifying and classifying spam emails. This testing phase presents several challenges and complexities, including:



Thanks for your subscription to Ringtone UK your mobile will be charged â£5/month Please confirm by replying YES or NO. If you reply NO you will not be charged

Spam

Data Variability: Real-world email data can be highly variable and dynamic, posing challenges for model generalization. The models must adapt to diverse email content, languages, and sender behaviours.



Is that seriously how you spell his name?

Not Spam

False Positive and False Negative Rates: The classifier's performance must be evaluated in terms of false positive (legitimate emails incorrectly classified as spam) and false negative (spam emails incorrectly classified as legitimate) rates. Balancing these rates is crucial for minimizing user inconvenience and ensuring effective spam filtering.

Feedback Mechanism: A feedback mechanism is implemented to collect user feedback on the accuracy of email classifications. This feedback loop helps improve the classifier's performance over time by incorporating user corrections and adjustments.

Performance in Simulation:

In cases where real-world testing poses challenges, simulation environments can be used to validate the classifier's performance. Simulated email datasets, comprising a diverse range of spam and legitimate emails, are used to assess the models' accuracy and reliability. This simulation-based testing provides insights into the classifier's behavior under controlled conditions and allows for thorough performance evaluation.

4.2.3. Model Validation and Testing

Model validation and testing are critical steps in ensuring the accuracy and reliability of the email spam classifier. In the context of the project, the following aspects are considered:

Validation Data:

Distinct from the training data, validation data is used to assess the predictive models' performance and generalization capabilities. The validation dataset comprises a representative sample of email data, including both spam and legitimate emails, sourced from diverse sources to mimic real-world scenarios accurately.

Testing Procedures:

During model testing, emails from the validation dataset are fed into the deployed models, and predictions are generated based on the classifier's decision boundaries. These predictions are compared against the actual labels (spam or legitimate) to evaluate the models' accuracy, precision, recall, and F1-score.

Testing Against Historical Data:

Historical email data, comprising past instances of spam and legitimate emails, is used to validate the models' performance. By applying the deployed models to historical email datasets, the effectiveness of the classifier in identifying past instances of spam is assessed, providing insights into its performance over time.

Testing in a Controlled Environment:

Controlled testing involves setting up a controlled email environment to assess the models' performance under specific conditions. This may include creating simulated email datasets with known characteristics (e.g., varying levels of spam content, different sender profiles) to evaluate how the models respond to different scenarios.

By addressing these aspects within the context of the "Email Spam Classifier using ML Model with NLP Techniques" project, the model performance evaluation aligns with the project's objective of accurately identifying and classifying spam emails using advanced techniques and models.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The "Email Spam Classifier using ML Model with NLP Techniques" project is a pivotal initiative in the realm of modern email security, aiming to bolster individuals' and organizations' defences against the ever-evolving landscape of email spam. The primary objective of this project is to leverage artificial intelligence, machine learning, and advanced data analysis techniques to predict and proactively identify potential instances of email spam, thereby enhancing email security and user experience.

Upon thorough examination of the project's core components, it becomes evident that its success hinges on its ability to harness vast volumes of data from diverse sources, including email content, metadata, user behaviors, and known spam patterns. By amalgamating various data sources and employing advanced predictive models, the project endeavors to predict and preemptively identify instances of email spam, thereby empowering users and organizations to safeguard their email communications effectively.

This project holds significant importance as it can significantly enhance an organization's email security posture and streamline their ability to anticipate, identify, and mitigate potential instances of email spam. By leveraging predictive analytics and pattern recognition, security professionals and organizations can stay one step ahead of malicious actors, detecting suspicious email patterns, anomalous behaviors, and indicators that foreshadow potential spam emails.

Furthermore, the project underscores the criticality of ethical considerations and data privacy concerns in the realm of email security. It emphasizes the importance of adhering to regulations and implementing responsible data handling practices to protect individuals' privacy and bolster email security.

As the proliferation of sophisticated email spam threats continues unabated, the development and deployment of predictive models and threat indicators assume increasing significance. Through proactive measures and the application of insights derived from the project, organizations can fortify their defenses against the dynamic landscape of email spam threats. In addition to safeguarding email communications, the project's proactive approach to identifying and thwarting spam emails contributes to the broader goal of fostering a more secure and resilient digital communication environment.

In summation, the "Email Spam Classifier using ML Model with NLP Techniques" project represents a pivotal and forward-thinking endeavor in the realm of email security, aimed at enhancing email security, protecting user privacy, and fortifying organizations' defenses against email spam threats.

5.2. Future Work

The "Email Spam Classifier using ML Model with NLP Techniques" project lays a solid foundation for enhancing email security and mitigating the impact of spam emails. However, there are several avenues for future work and research that can further refine and extend the capabilities of the classifier system. Some potential areas for future work include:

1. **Enhanced Feature Engineering:** Explore and incorporate additional features derived from email content, metadata, sender reputation, and user behaviors to improve the classifier's accuracy and robustness. Experiment with advanced feature extraction techniques, such as word embeddings, semantic analysis, and sentiment analysis, to capture deeper insights from email text.
2. **Deep Learning Architectures:** Investigate the application of deep learning architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer models, for email spam classification. Explore how these architectures can leverage the

hierarchical structure and sequential nature of email data to capture complex patterns and relationships.

3. **Transfer Learning and Pretrained Models:** Explore the use of transfer learning and pretrained language models, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pretrained Transformer), to leverage large-scale text corpora and domain-specific knowledge for email spam classification. Fine-tune these pretrained models on email data to adapt them to the specific characteristics of spam detection.
4. **Active Learning Strategies:** Investigate active learning strategies to intelligently select informative samples for model training and annotation, thereby reducing the manual labeling effort and improving the efficiency of the classifier. Explore techniques such as uncertainty sampling, query-by-committee, and diversity sampling to prioritize the acquisition of labeled data that maximizes the classifier's performance gains.
5. **Imbalanced Data Handling:** Develop techniques to address class imbalance in the email dataset, where the number of spam emails may be significantly lower than legitimate emails. Explore oversampling, undersampling, and hybrid sampling approaches to balance the class distribution and prevent the classifier from being biased towards the majority class.
6. **Dynamic Model Adaptation:** Implement mechanisms for dynamic model adaptation and continuous learning to adapt the classifier to evolving spam email patterns and tactics. Explore online learning algorithms and incremental training strategies to update the classifier's parameters in real-time based on incoming email data and user feedback.
7. **Multi-level Ensemble Approaches:** Investigate ensemble learning approaches that combine predictions from multiple base classifiers at different levels of abstraction or feature representations. Explore

techniques such as stacking, boosting, and hierarchical ensembles to harness the diversity of individual classifiers and improve overall classification performance.

8. **User Feedback Integration:** Develop mechanisms for integrating user feedback into the classifier system to iteratively refine and improve its performance over time. Implement feedback loops that allow users to provide corrections and annotations on misclassified emails, enabling the classifier to adapt to user preferences and evolving email patterns.
9. **Scalability and Efficiency:** Address scalability and efficiency challenges associated with processing large volumes of email data in real-time or batch mode. Explore distributed computing frameworks, stream processing technologies, and cloud-based solutions to enable parallelized and scalable model inference and deployment.
10. **Evaluation on Diverse Datasets:** Conduct comprehensive evaluations of the classifier system on diverse datasets sourced from different email providers, domains, languages, and regions. Assess the generalization and robustness of the classifier across various email environments and demographics to ensure its effectiveness in real-world scenarios.

REFERENCES

[1] A Sharaff and Srinivasarao U (2020), ”Towards classification of email through selection of informative features,” First International Conference on Power, Control and Computing Technologies (ICPC2T), Raipur, India, pp. 316-320, DOI: 10.1109/ICPC2T48082.2020.9071488.

[2] Navaney, P., Dubey, G., Rana, A. (2018). ”SMS Spam Filtering Using Supervised Machine Learning Algorithms.” 2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence).

[3]] J. Marseline K.S and Nandhini S (2020), ”Performance Evaluation of Machine Learning Algorithms for Email Spam Detection,” International

Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, pp. 1-4, DOI: 10.1109/icETITE47903.2020.312.

[4] Jyoti Prakash Singh, Pradeep Kumar Roy and Snehasish Banerjee (2019), “Deep learning to filter SMS spam,” Future Generation computer Systems, vol .102, pp. 524-533, DOI: 10.1016/j.future.2019.09.001

[5] Oguz Emre Kural and Sercan Demirci (2020), ”Comparison of Term Weighting Techniques in Spam SMS Detection,” 28th Signal Processing and Communications Applications Conference (SIU), Gaziantep, Turkey, 2020, pp. 1-4, DOI: 10.1109/SIU49456.2020.9302315..

[6] R. Abinaya, P. Naveen and B. Niveda E (2020), ”Spam Detection On Social Media Platforms”, 7th International Conference on Smart Structures and Systems (ICSSS), Chennai, India, pp. 1-3, DOI: 10.1109/ICSSS49621.2020.9201948.

[7] S. B. Rathod and T. M. Pattewar, “A comparative performance evaluation of content based spam and malicious URL detection in E-mail”, 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), Bhubaneswar, 2015, pp. 49-54, doi: 10.1109/CGVIS.2015.7449891.

[8] Wei Hu, Jinglong Du, and Yongkang Xing, “Spam Filtering by Semantics-based Text Classification”, 8th International Conference on Advanced Computational Intelligence Chiang Mai, Thailand; February 14-16, 2016

[9] Crawford, M., Khoshgoftaar, T.M., Prusa, J.D. et al. ,“Survey of review spam detection using machine learning techniques”, Journal of Big Data 2, 23 (2015). https://doi.org/10.1186/s40537- 015-0029-9

[10] Vlad Sandulescu, Martin Ester “Detecting Singleton Review Spammers Using Semantic Similarity”, WWW ’15 Companion Proceedings of the 24th International Conference on World Wide Web, 2015, p.971-976 10.1145/2740908.2742570

[11] Cheng Hua Li, Jimmy Xiangji Huang “Spam filtering using semantic similarity approach and adaptive BPNN”, Neurocomputing Journal, Elsevier, https://doi.org/10.1016/j.neucom.2011.09.036

[12] Krishnan Kannoorpatti, Asif Karim , Sami Azam, BharanidharanSanmugam, ”on A Comprehensive Survey for Intelligent Spam Email Detection,” IEEE Journal of Computational Intelligence, 2015.

[13] Zainal K, Sulaiman NF, Jali MZ, “An Analysis of Various Algorithms For Text Spam Classification and Clustering Using RapidMiner and Weka”, (IJCSIS) International Journal of Computer Science and Information Security, Vol. 13, No. 3, March 2015

[14] S.M.Lee, D.S.Kim, J.H.Kim, J.S.Park, “Spam Detection Using Feature Selection and Parameter Optimization”, 2010 International Conference on

Complex, Intelligent and Software Intensive Systems, DOI 10.1109/CISIS.2010.116

[15] E. Markova, T. Bajto ´ s, P. Sokol and T. M ˇ eze ´ sov ˇ a, “Classification of ´ malicious emails”, 2019 IEEE 15th International Scientific Conference on Informatics, Poprad, Slovakia, 2019, pp. 000279-000284, doi: 10.1109/Informatics47936.2019.9119329.

[16] M. S. Swetha and G. Sarraf, “Spam Email and Malware Elimination employing various Classification Techniques”, 2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT), Bangalore, India, 2019, pp. 140-145, doi: 10.1109/RTEICT46194.2019.9016964.

[17] S. Nandhini and D. J. Marseline.K.S, “Performance Evaluation of Machine Learning Algorithms for Email Spam Detection”, 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 2020, pp. 1-4, doi: