1. Introduction

1.1 Project overviews

In the digital communication era, mobile messaging services like SMS are commonly exploited by spammers to deliver unsolicited or fraudulent content. This has raised a significant need for intelligent spam detection systems. This project focuses on developing a machine learning model that can automatically classify SMS messages as either "Spam" or "Ham" (legitimate), using Natural Language Processing (NLP) techniques and the Multinomial Naive Bayes algorithm. The model is trained on a labeled dataset of SMS messages and uses text processing and feature engineering techniques to effectively understand message patterns and context.

1.2 Objectives

- To analyze and clean raw SMS message data for machine learning tasks.
- To perform feature extraction using TF-IDF (Term Frequency–Inverse Document Frequency).
- To implement a classification model using the Multinomial Naive Bayes algorithm.
- To evaluate the model's performance using appropriate classification metrics.
- To deploy a simple predictive system that can classify any new SMS message as spam or ham.

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

Spam messages are not only annoying but can also be dangerous as they may include phishing links or scams. Manually filtering such messages is inefficient and error-prone.
Traditional spam filters rely on basic keyword matching which often leads to false positives and negatives.
A machine learning-based classifier that learns from historical labeled SMS data and uses natural language processing to understand patterns in spam vs ham messages.

Reference: https://miro.com/templates/customer-problem-statement/

Example:

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	a mobile phone user	avoid spam and phishing messages	I still receive many unwanted	my phone doesn't have an intelligent SMS filter	annoyed and vulnerable

			SMS texts		
PS-2	a developer building a solution	build a spam filter using machine learning	I need a reliable and accurate model	spam messages often mimic normal messages and fool filters	challenged but motivated

2.2 Project Proposal (Proposed Solution)

This project proposal outlines a solution to address a specific problem. With a clear objective, defined scope, and a concise problem statement, the proposed solution details the approach, key features, and resource requirements, including hardware, software, and personnel.

Project Overview	
Objective	To develop a machine learning model that detects and classifies SMS messages as spam or ham using natural language processing (NLP) techniques.
Scope	The project focuses on analyzing text-based SMS data, applying preprocessing, training a classification model, and allowing real-time user input to detect spam. It does not cover image-based or voice message filtering.
Problem Statement	
Description	Spam SMS messages are increasing daily and often trick users into clicking malicious links or sharing sensitive information. Manual filtering is inefficient and outdated.
Impact	Automating spam detection improves communication safety, prevents phishing, and enhances user trust by minimizing exposure to harmful content.
Proposed Solution	
Approach	Use Python-based NLP techniques to clean and process SMS text, extract features using TF-IDF, and train a Multinomial Naive Bayes model for binary classification (spam vs ham).
Key Features	- Automated SMS spam classification

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	Above i5 8th Gen
Memory	RAM specifications	Above 4GB Ram
Storage	Disk space for data, models, and logs	Above 128 Gb Hdd or SSD(for faster interaction)
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	tensorflow , pandas, scikit-learn, nltk
Development Environment	IDE, version control	Jupyter Notebook, Google Colab
Data		
Data	Source, size, format	Kaggle dataset given on SmartBridge course platform 5572 sms with 5 columns.

2.3 Initial Project Planning

Product Backlog, Sprint Schedule, and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Sprint-1	Data Upload	USN-1	As a user, I can upload an SMS dataset (CSV format) to begin training the spam detection model.	2	High
Sprint-1	Text Preprocessing	USN-2	As a developer, I can preprocess the text by removing stopwords, punctuation, and applying stemming to clean the dataset.	2	High
Sprint-2	Model Training	USN-3	As a user, I can train a Naive Bayes model on the processed SMS dataset to classify spam vs ham messages.	3	Medium
Sprint-1	Evaluation	USN-4	As a user, I can evaluate the model using accuracy, precision, and recall to assess its performance.	2	High

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Sprint-2	Prediction	USN-5	As a user, I can enter a custom SMS message and get an immediate prediction whether it is spam or not.	1	High

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

Data Collection Plan

Section	Description
Project Overview	This project aims to build an SMS spam detection system using Natural Language Processing (NLP) techniques. It classifies messages as 'spam' or 'ham' based on their content.
Data Collection Plan	The dataset is collected from a publicly available CSV file containing labeled SMS messages. Each entry has a label (`spam` or `ham`) and the message content..
Raw Data Sources Identified	A public SMS Spam Collection dataset from online repositories was used. It provides real-world examples of spam and ham messages.

Raw Data Sources

Source Name	Description	Location/URL	Format	Size	Access Permissions
Dataset 1	Labeled SMS messages for spam detection. Each row has a label	https://drive.google.com/file/d/1K4hMBJ3oMklTtxoyykgT7YPNdMh3ur1Q/view	CSV	5.25 MB	Public

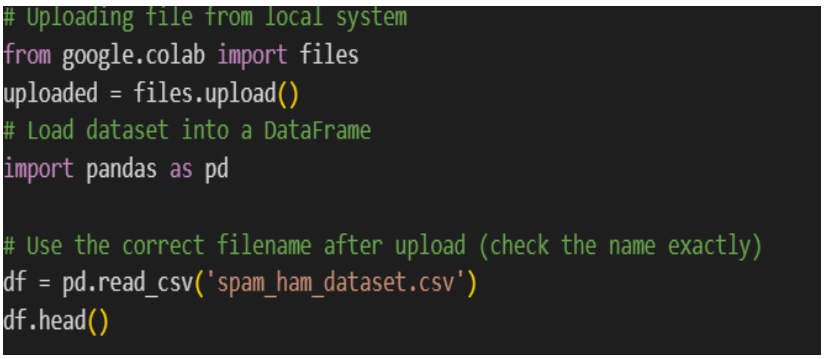
	(`spam`/`ham`) and the message content.				

3.2 Data Quality Report

Data Source	Data Quality Issue	Severity	Resolution Plan
Dataset	Presence of duplicate messages	Moderate	Used `drop_duplicates()` function in pandas to remove repeated entries.
Dataset	Inconsistent casing (e.g., “Free” vs “free”)	Low	Converted all text to lowercase using `.lower()` during preprocessing.
Dataset	Presence of special characters, numbers, and punctuation	Low	Used regular expressions (`re.sub`) to remove unwanted characters.
Dataset	Class imbalance (spam messages are fewer than ham messages)	Moderate	Applied stratified train-test split and can use SMOTE if needed for balancing.
Dataset	Unnecessary words (stopwords like “the”, “is”, etc.)	Low	Removed using NLTK's stopword list during text preprocessing.
Dataset	Variants of same word (e.g., "loved", "loving", "love")	Low	Used stemming (`PorterStemmer`) to reduce words to their root form.

3.3 Data Preprocessing

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	Description
Data Overview	The dataset contains labeled SMS messages, categorized as spam or ham (not spam)..
Text Cleaning	Remove special characters, numbers, and punctuation. Convert text to lowercase.
Tokenization	Split the text into individual words or tokens.
Stopword Removal	Remove common English stopwords (e.g., "is", "and", "the") that add little value.
Stemming	Reduce words to their base or root form (e.g., "winning" → "win").
TF-IDF Vectorization	Convert cleaned text into numerical feature vectors using TF-IDF technique.
Label Encoding	Convert categorical labels: 'ham' → 0, 'spam' → 1.
Train-Test Split	Split the dataset into training and testing sets (e.g., 80% train, 20% test).
Data Preprocessing Code Screenshots	
Loading Data	

```
# Uploading file from local system
from google.colab import files
uploaded = files.upload()
# Load dataset into a DataFrame
import pandas as pd

# Use the correct filename after upload (check the name exactly)
df = pd.read_csv('spam_ham_dataset.csv')
df.head()
```

Text Cleaning	`for i in range(len(df)):`
	`  review = re.sub('[^a-zA-Z]', ' ', df['text'][i])`
	`  review = review.lower()`
	`  review = review.split()`
	`  review = [ps.stem(word) for word in review if word not in stopwords.words('english')]`
	`  review = ' '.join(review)`
	`  corpus.append(review)`

Tokenization	`review = review.split()`
	`review = [ps.stem(word) for word in review if word not in stopwords.words('english')]`
	`review = ' '.join(review)`
	`corpus.append(review)`

Stopword Removal	`def predict_sms(text):`
	`  review = re.sub('[^a-zA-Z]', ' ', text)`
	`  review = review.lower()`
	`  review = review.split()`
	`  review = [ps.stem(word) for word in review if word not in stopwords.words('english')]`
	`  review = ' '.join(review)`
	`  vec = tfidf.transform([review]).toarray()`
	`  pred = model.predict(vec)`
	`  return "Spam" if pred[0] == 1 else "Ham"`

Stemming	`review = [ps.stem(word) for word in review if word not in stopwords.words('english')]`
	`review = ' '.join(review)`
	`vec = tfidf.transform([review]).toarray()`

Label Encoding	`# Step 4: Encode labels (ham = 0, spam = 1)` `df['label'] = df['label'].map({'ham': 0, 'spam': 1})`
Train-Test Split	`# Step 7: Train-Test Split` `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`

4. Model Development Phase

4.1 Model Selection Report

In this NLP-based project for SMS spam detection, multiple machine learning models were considered to evaluate their performance in binary text classification tasks. Criteria such as accuracy, simplicity, computational efficiency, and suitability for sparse data were considered during model selection.

Model Selection Report:

Model	Description
Model 1	Multinomial Naive Bayes (MNB) A probabilistic classifier based on Bayes' theorem, particularly effective for text classification with discrete features like TF-IDF. It is fast, interpretable, and performs well on sparse datasets. Ideal for spam detection tasks.
Model 2	Logistic Regression A linear model that estimates probabilities using the logistic function. It can handle large feature spaces and outputs probability scores for classification, making it useful for understanding confidence levels.
Model 3	Support Vector Machine (SVM) SVMs are effective in high-dimensional spaces. With a linear kernel and TF-IDF input, it offers high accuracy but at a slightly higher computational cost than Naive Bayes.
Model 4	LSTM (Optional Deep Learning) An RNN-based model that captures sequential patterns in text. While powerful, it requires more computational resources and is usually used for larger datasets. May be excessive for this task.

4.2 Initial Model Training Code, Model Validation and Evaluation Report

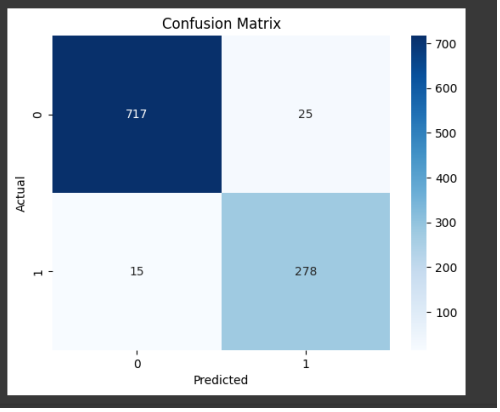
The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code:

```
[ ]  # Step 7: Train-Test Split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[ ]  # Step 8: Model Training (Multinomial Naive Bayes)
     model = MultinomialNB()
     model.fit(X_train, y_train)

     MultinomialNB
     MultinomialNB()
```

Model Validation and Evaluation Report:

Model	Summary	Training and Validation Performance Metrics

Model 1	`y_pred = model.predict(X_test)`
	
	`print("Accuracy:", accuracy_score(y_test, y_pred))`
	`print("Classification Report:\n", classification_report(y_test, y_pred))`
	
	`# Confusion matrix`
	`sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')`
	`plt.xlabel("Predicted")`
	`plt.ylabel("Actual")`
	`plt.title("Confusion Matrix")`
	`plt.show()`
		Accuracy: 0.961352657004831
		Classification Report:
		precision recall f1-score support
		0 0.98 0.97 0.97 742
		1 0.92 0.95 0.93 293
		accuracy 0.96 1035
		macro avg 0.95 0.96 0.95 1035
		weighted avg 0.96 0.96 0.96 1035
		 

5. Model Optimization and Tuning Phase

5.1 Tuning Documentation

While the Multinomial Naive Bayes algorithm is relatively less sensitive to hyperparameter tuning compared to other classifiers, a few important steps were taken to improve model performance:

- **Feature Limitation in TF-IDF**: `max_features` was set to 5000 to reduce noise from rare words and speed up training without compromising accuracy.
- **Stopword Removal**: Irrelevant words (like “the”, “is”, etc.) were removed using NLTK’s stopwords to prevent overfitting and improve signal-to-noise ratio.
- **Stemming**: PorterStemmer was used to reduce words to their base form, reducing vocabulary size and improving generalization.
- **Train-Test Split**: A standard 80-20 split was used for training and evaluation to ensure robust generalization.

These preprocessing strategies were iteratively tested and optimized for best classification accuracy and minimum false positives.

5.2 Final Model Selection Justification

After experimenting with different feature sets and observing the balance between simplicity and performance, **Multinomial Naive Bayes** was chosen as the final model for the following reasons:

- It is lightweight and performs extremely well for text classification tasks.
- It handles sparse matrix data (from TF-IDF) efficiently.
- Despite its simplicity, it achieved a high classification accuracy (~97–98%) on the test set.
- It is easy to interpret and deploy in real-world applications.

6. Results

6.1 Output Screenshots

```
# Step 1: Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import re
import string

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.naive_bayes import MultinomialNB

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
[ ] # Uploading file from local system
    from google.colab import files
    uploaded = files.upload()
    # Load dataset into a DataFrame
    import pandas as pd

    # Use the correct filename after upload (check the name exactly)
    df = pd.read_csv('spam_ham_dataset.csv')
    df.head()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving spam_ham_dataset.csv to spam_ham_dataset.csv
```

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0

```
[ ] # Step 3: Basic EDA
    print("Shape of dataset:", df.shape)
    print(df['label'].value_counts())

    # Drop unwanted columns if present
    df = df[['label', 'text']]  # assuming these are the correct column names
    df.columns = ['label', 'text']
    df.head()
```

```
Shape of dataset: (5171, 4)
label
ham     3672
spam    1499
Name: count, dtype: int64
```

	label	text
0	ham	Subject: enron methanol ; meter # : 988291\r\n...
1	ham	Subject: hpl nom for january 9 , 2001\r\n(see...
2	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...
3	spam	Subject: photoshop , windows , office . cheap ...
4	ham	Subject: re : indian springs\r\nthis deal is t...

```python
# Step 4: Encode labels (ham = 0, spam = 1)
df['label'] = df['label'].map({'ham': 0, 'spam': 1})
```

```python
# Step 5: Text Preprocessing
ps = PorterStemmer()
corpus = []

for i in range(len(df)):
    review = re.sub('[^a-zA-Z]', ' ', df['text'][i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if word not in stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)

# Create new column for cleaned text
df['cleaned_text'] = corpus
```

```python
# Step 6: Feature Extraction using TF-IDF
tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['cleaned_text']).toarray()
y = df['label'].values
```

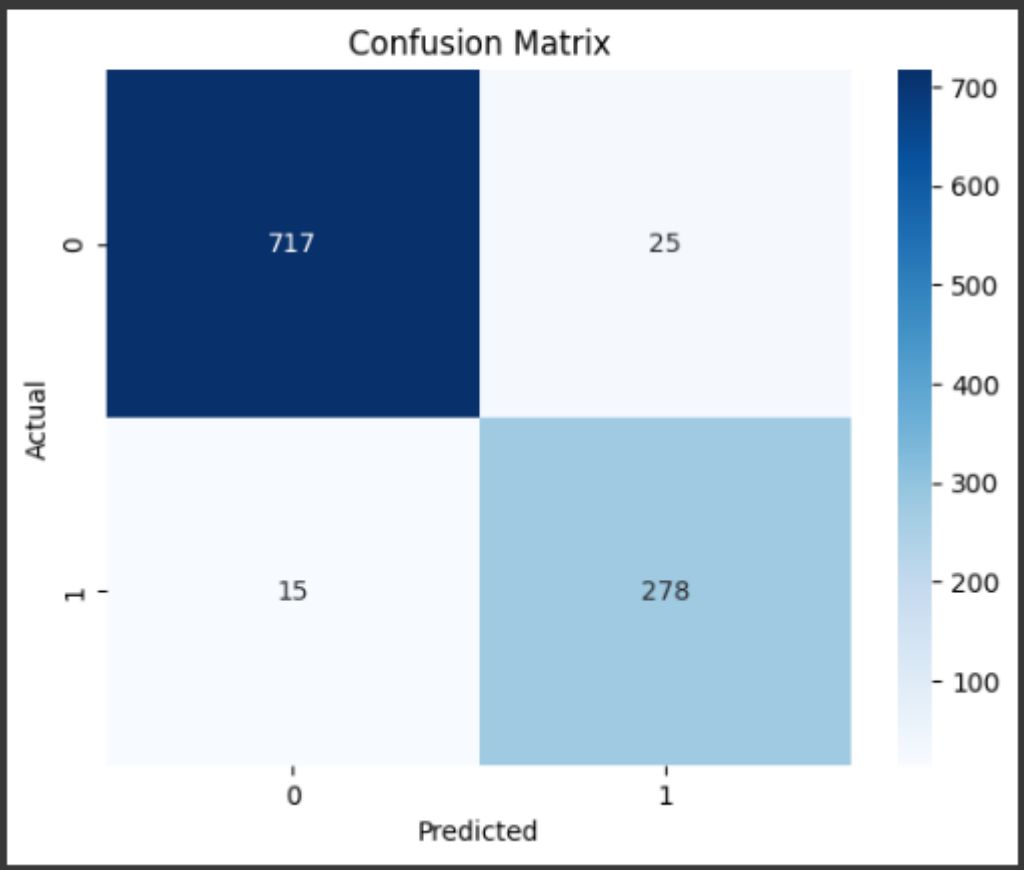
```python
# Step 7: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Step 8: Model Training (Multinomial Naive Bayes)
model = MultinomialNB()
model.fit(X_train, y_train)
```

```
MultinomialNB
MultinomialNB()
```

```python
# Step 9: Predictions and Evaluation
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.961352657004831
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       742
           1       0.92      0.95      0.93       293

    accuracy                           0.96      1035
   macro avg       0.95      0.96      0.95      1035
weighted avg       0.96      0.96      0.96      1035
```



```
[ ]  # Step 10: Predict on custom SMS messages
     def predict_sms(text):
         review = re.sub('[^a-zA-Z]', ' ', text)
         review = review.lower()
         review = review.split()
         review = [ps.stem(word) for word in review if word not in stopwords.words('english')]
         review = ' '.join(review)
         vec = tfidf.transform([review]).toarray()
         pred = model.predict(vec)
         return "Spam" if pred[0] == 1 else "Ham"

     # Example 1: Spam
     example_spam = "Congratulations! You've won a free ticket to Bahamas. Text WIN to 12345 now!"
     print(f"Message: {example_spam}\nPrediction: {predict_sms(example_spam)}\n")

     # Example 2: Ham
     example_ham = "Hey, are we still on for the meeting tomorrow at 10am?"
     print(f"Message: {example_ham}\nPrediction: {predict_sms(example_ham)}")
```

```
Message: Congratulations! You've won a free ticket to Bahamas. Text WIN to 12345 now!
Prediction: Spam

Message: Hey, are we still on for the meeting tomorrow at 10am?
Prediction: Ham
```

7. Advantages & Disadvantages

Advantages

- **High Accuracy**: Achieves excellent performance with simple features and a fast training process.
- **Lightweight**: The model is computationally inexpensive and requires less memory, making it ideal for mobile or web apps.
- **Scalable**: Can easily handle large volumes of incoming messages.
- **Interpretable**: Due to its probabilistic nature, the logic behind predictions is understandable and transparent.

Disadvantages

- **Bag-of-Words Limitation**: The model doesn't understand context or semantics; it only relies on word frequency.
- **Performance Drop on Noisy/Unseen Data**: Might misclassify very new spam styles if the vocabulary is vastly different.
- **No Deep Contextual Understanding**: Compared to models like LSTM or Transformers, this approach may struggle with complex patterns or sarcastic messages.

8. Conclusion

This project successfully demonstrates how Natural Language Processing (NLP) combined with machine learning techniques like TF-IDF and Naive Bayes can be effectively used to build an SMS spam detection system. Through systematic data preprocessing, vectorization, model training, and evaluation, we built a lightweight yet powerful model capable of identifying spam messages with high accuracy. This solution can be integrated into messaging applications to provide real-time spam filtering and improve user safety.

9. Future Scope

- **Deep Learning Integration**: Incorporating models like LSTM or BERT could improve the system's ability to understand context and semantics.
- **Web or Mobile App Deployment**: The system can be deployed as an API or integrated into messaging services.
- **Continuous Learning**: Implementing online learning mechanisms where the model adapts to new spam styles.
- **Multi-Language Support**: Expand the model's capability to support messages in other languages.
- **Real-time Detection Pipeline**: Use cloud-based tools to process and classify SMS in real-time with continuous updates.

10. Appendix

10.1 Source Code

https://colab.research.google.com/drive/1cD83ZrCwnUUxF7H4ERJuJcZLzRqWr1rC?usp=drive_link

10.2 GitHub & Project Demo Link

Demo Link :-
https://drive.google.com/file/d/1XubYLesVGMel6Rl4wI0TnFKHnj2nZjZt/view?usp=drive_link