Project Title :

Building a Smarter AI-Powered Spam Classifier

Phase 5 : Project Documentation and Submission

DOCUMENTATION:-

PROBLEM STATEMENT:

Project Overview:

The central aim of this project is to conceptualize and construct an AI-powered spam classifier employing the capabilities of Natural Language Processing (NLP) and machine learning techniques. The fundamental goal is to craft a resilient and precise system with the capacity to distinguish spam from legitimate messages across a spectrum of communication channels, including emails and text messages. The project's mission is to enhance communication security, minimize false positives, and elevate the accuracy of spam detection.

DESIGN THINKING PROCESS:-

1.Data Collection:

Objective:

- Obtain a suitable dataset comprising labeled instances of both spam and non-spam messages.

Actions:

- Identify potential data sources, encompassing platforms such as Kaggle, academic repositories, and publicly accessible datasets.
- Verify that the dataset encompasses a sufficient volume of labeled spam and non-spam messages to enable comprehensivemodel training and evaluation.

2.Data Preprocessing:

Objective:

- Prepare textual data for analytical purposes by engaging in data refinement and structuring.

Tasks:

- Execute data cleansing to eliminate specialcharacters, punctuation marks, and irrelevant symbols from the text corpus.
- Standardize the text by transforming it to lowercase to ensure uniformity.
- Apply tokenization to break down the text into individual words or tokens for subsequent analysis.

Data preprocessing is the process of converting raw data into a format that is understandable and usable. It is a crucial step in any Data Science project to carry out an efficient and accurate analysis.

Data preprocessing involves:

- Cleaning, transforming, and integrating data.
- Checking for missing values, noisy data, and other inconsistencies.
- Improving the quality of the data .
- Making the data more suitable for the specific data mining task

3.Feature Extraction:

Objective:

- Convert textual data into numeric features conducive to machine learning.

Technique:

- Implement the TF-IDF (Term Frequency-Inverse Document Frequency) methodology to translate tokenized words into numerical values.

Actions:

- Employ TF-IDF vectorization to represent tokenized words as numerical attributes.
- Determine an appropriate threshold for the maximum number of features, guided by dataset dimensions and characteristics.

4.Model Selection:

Recommadations:

- Commence the project by selecting the Naive Bayes algorithm as the primary model. Naive Bayes, acclaimed for its simplicity and effectiveness in text classification, often proves to be a fitting choice for spam detection.
- If Naive Bayes meets the desired accuracy and performance criteria, it can serve as the principal model.

Actions (if necessary):

- In the event of Naive Bayes falling short of achieving the desired accuracy or encountering specific challenges, consider delving into alternative algorithms, such as Support Vector Machines (SVM) or deep learning (neural networks), based on the dataset's intricacy and performance benchmarks.

5.**Model Training and Evaluation:**

Objective:

- Train the chosen model and assess its performance.

Actions:

- Divide the dataset into training and testing subsets.
- Administer training to the selected model using the training data.
- Evaluate the model's performance employing pertinent metrics, including accuracy, precision, recall, and the F1-score.

- Utilize a confusion matrix to glean deeper insights into classification

outcomes.

6.**Iterative Improvement**:

Objective:

- Enhance the spam classifier iteratively through continuous refinement.

Strategies for Enhancement:

- Hyperparameter Tuning
- Feature Engineering
- Data Augmentation (if dataset size presents constraints)
- Regularization (to mitigate overfitting)
- Ongoing Monitoring and Integration of User Feedback

Actions:

- Fine-tune the model by optimizing hyperparameters to achieve peak performance.
- Explore feature engineering techniques to extract richer insights from the textual data.
- Delve into strategies for data augmentation, particularly if the dataset exhibits limitations in size.
- Implement regularization techniques, especially beneficial for intricate models such as deep learning.
- Maintain vigilant oversight of the classifier's performance in real-world scenarios and assimilate user feedback to drive ongoing enhancements.

Expected Deliverables:

- Upon the culmination of the project, the following deliverables are anticipated:
- A trained AI-powered spam classifier with the capability to accurately differentiate between spam and non-spam messages.
- A comprehensive of evaluation results, spotlighting the model's performance using pertinent metrics.
- Exhaustive documentation delineating the project's methodology, data provenance, data preprocessing steps, feature extraction approaches, and iterative enhancement strategies.
- A repository housing the implemented spam classifier, along with any essential scripts for datapreprocessing and model evaluation.

PHASES OF DEVELOPMENT:

Phase 1: Data Collection and Loading

Problem Statement:

Gather a dataset of SMS messages for spam classification.

Activities:

Collect a dataset of SMS messages labeled as spam and non-spam. The dataset may come from various sources or be downloaded from platforms like Kaggle. Load the dataset into a Pandas DataFrame using the read_csv function.

Output: A Pandas DataFrame containing the SMS dataset.

Phase 2: Exploratory Data Analysis (EDA)

Problem Statement:

Understand the dataset's characteristics and distribution.

Activities:

- Visualize the distribution of spam and non-spam messages using a bar plot.
- Analyze the distribution of message length and create a histogram to understand how message length varies by label.
- Create a scatter plot to explore the relationship between message length and labels.

Output: Data visualizations that provide insights into the dataset's characteristics and distribution.

Phase 3: Data Preprocessing

Problem Statement:

Prepare the data for machine learning by removing inconsistencies.

Activities:

Remove duplicate messages to avoid data redundancy.
Convert the text in the 'v2' column to lowercase to ensure uniformity.

Output: A preprocessed dataset with duplicate messages removed and text converted to lowercase.

Phase 4: Basic Data Processing Methods

Problem Statement:

Provide an overview of the preprocessed dataset.

Activities:

- Display the first five and last five rows of the dataset to provide a glimpse of the data.
- Share additional information about the dataset, including its shape (number of rows and columns), size, and presence of missing values.
- Provide descriptive statistics of the dataset.
- Analyze the number of unique values in the 'v1' column (label) and display value counts.
- List the columns present in the dataset.
- Save the preprocessed dataset to a CSV file for future use.

Output: A detailed overview of the preprocessed dataset and a saved CSV file.

Phase 5: Data Splitting

Problem Statement:

Prepare the dataset for model training and evaluation by splitting it into training and testing sets.

Activities:

- Use the train_test_split function from Scikit-Learn to create training and testing sets.

Output: Training and testing datasets.

Phase 6: Text Vectorization using TF-IDF

Problem Statement:

Convert text data into numerical features suitable for machine learning models.

Activities:

- Create TF-IDF vectors from text data using the TfidfVectorizer from Scikit-Learn. This involves removing stop words, tokenizing text, and calculating TF-IDF scores.

Output: TF-IDF vectors for the training and testing sets.

Phase 7: Model Training and Evaluation

Problem Statement:

Develop and evaluate a machine learning model for spam classification.

Activities:

- Train a Multinomial Naive Bayes model using the TF-IDF vectors.
- Make predictions on the test data.
- Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score.

Output: Evaluation metrics and the trained machine learning model.

Phase 8: Additional Metrics

Problem Statement:

Further assess the model's performance using cross-validation, confusion matrix, and a classification report.

Activities:

- Perform cross-validation to assess the model's generalization performance.
- Calculate the confusion matrix to understand the model's true positives, true negatives, false positives, and false negatives.
- Generate a classification report, which includes metrics like precision, recall, F1 score, and support.

Output: Cross-validation scores, confusion matrix, and a classification report to provide additional insights into the model's performance.

These phases together cover the complete development cycle for spam classification, from data collection and preprocessing to model training and evaluation, including additional performance metrics.

DATASET USED, DATA PREPROCESSING STEPS, AND FEATURE EXTRACTION TECHNIQUES:-

Dataset Description:

- **Name:** The dataset is named "AI_Phase5.csv."
- **Source:** It is not explicitly mentioned, but it could have been obtained from various sources or datasets available for spam classification, such as the UCI Machine Learning Repository or Kaggle.
- **Data Format:** The dataset is provided in a CSV (Comma-Separated Values) format.
- **Features:** The dataset contains two main columns:
    1. "v1": This column contains labels indicating whether a message is "spam" or "ham" (non-spam).
    2. "v2": This column contains the text of SMS messages.

Data Preprocessing Steps:

1. **Duplicate Removal:** The dataset preprocessing phase involves removing duplicate SMS messages to avoid redundancy and to ensure that each message is unique.
2. **Text Lowercasing:** All the text data in the "v2" column is converted to lowercase. This process ensures uniformity in the text and avoids issues related to case sensitivity.

Feature Extraction Techniques:

- In this code, the primary feature extraction technique employed is **TF-IDF (Term Frequency-Inverse Document Frequency) vectorization**. TF-IDF is a numerical representation of text data that captures the importance of words in a document

relative to a collection of documents (corpus). Here are the steps for TF-IDF feature extraction:

1. **Text Tokenization:** The text data is tokenized into individual words. This process breaks down each message into its constituent words or tokens.
2. **Stop Word Removal:** Common English stop words are removed from the tokens. Stop words are words that do not carry significant meaning (e.g., "the," "is," "in") and are often removed to focus on more meaningful words.
3. **TF-IDF Calculation:** The TF-IDF scores for each word in the document are calculated. TF-IDF combines two measures:
    - **Term Frequency (TF):** The number of times a word appears in a document. It reflects the importance of a word within that document.
    - **Inverse Document Frequency (IDF):** The inverse of the number of documents in which a word appears. It quantifies the rarity of a word across the corpus.
4. **Vectorization:** The TF-IDF scores are used to create numerical feature vectors. Each document (SMS message) is represented by a vector, where each component corresponds to a word and its TF-IDF score.

These feature vectors are then used as input data to train and evaluate a Multinomial Naive Bayes model for spam classification. The TF-IDF representation helps capture the importance of words in distinguishing

between spam and non-spam messages, making it a common choice for text classification tasks.

MACHINE LEARNING ALGORITHM, MODEL TRAINING, AND EVALUATION METRICS:-

Machine Learning Algorithm:

- **Chosen Algorithm:**

  In the provided code, a Multinomial Naive Bayes (NB) classifier is chosen as the machine learning algorithm. Naive Bayes is a probabilistic algorithm that is particularly well-suited for text classification tasks like spam detection. The "Multinomial" variant of Naive Bayes is commonly used when dealing with discrete data like word counts in text documents**.**

Model Training:

- **Training Data:** The dataset is divided into training and testing sets using the train_test_split function. The training set is used to train the machine learning model, and the testing set is used to evaluate the model's performance.
- **Feature Extraction:** The text data is preprocessed, cleaned, and transformed into numerical feature vectors using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique. The TF-IDF vectors capture the importance of words in distinguishing spam from non-spam messages
- **Model Training:** The Multinomial Naive Bayes model is trained on the training data. During training, the model learns the probabilistic relationship between the TF-IDF features and the corresponding spam or non-spam labels. It estimates the likelihood of observing a particular set of features given the class label.

Evaluation Metrics:

- **Accuracy**: Accuracy measures the overall correctness of the model's predictions. It is the ratio of correctly classified instances to the total instances in the testing set.

- **Precision:** Precision is the ratio of true positives (correctly predicted spam) to the total predicted positives (true positives + false positives). It measures the ability of the model to make precise spam predictions.
- **Recall:** Recall is the ratio of true positives to the total actual positives (true positives + false negatives). It measures the model's ability to identify all actual spam messages.
- **F1 Score**: The F1 Score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is particularly useful when there is an imbalance between the two classes (spam and non-spam).

Cross-Validation:

- Cross-validation is used to assess the performance of the model more robustly. In the code, a 5-fold cross-validation is performed. It involves splitting the training data into five subsets, using four for training and one for validation iteratively. This helps in estimating the model's generalization performance and ensures that the model is not overfitting to the training data.

The choice of Multinomial Naive Bayes is suitable for text classification tasks, and the TF-IDF vectorization technique is a good way to represent text data numerically. The evaluation metrics provide a well-rounded view of the model's performance, taking into account both precision and recall, and cross-validation helps ensure the model's reliability**.**

1. **Deep Learning Models:** Innovative spam classifiers may employ deep learning models, such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs), to capture complex patterns in text data. These models have shown promising results in natural language processing tasks.
2. **Word Embeddings:** Advanced techniques like Word2Vec or GloVe pre-trained word embeddings can be used to represent words as dense vectors. These embeddings capture semantic relationships among words and can enhance the model's understanding of the text.
3. **Ensemble Methods:** Combining multiple models, such as Naive Bayes, Support Vector Machines (SVM), and deep learning models, into an ensemble can improve overall performance. Techniques like stacking or bagging can be used for this purpose.
4. **Anomaly Detection:** Unsupervised anomaly detection techniques, like one-class SVM or Isolation Forest, can be used to detect unusual patterns in messages, which may indicate spam.
5. **Active Learning:** Innovative approaches may incorporate active learning strategies to reduce the manual labeling effort required for building a labeled dataset. The model selects the most uncertain instances for manual labeling, optimizing the training process.

6. **Temporal Analysis:** Some spam messages exhibit temporal patterns. Innovative classifiers can analyze message timestamps to identify spam messages that arrive at unusual times.
7. **Graph-Based Models:** Leveraging social network or communication graph structures can be a powerful technique to detect spam. Analyzing connections and interactions between users can help identify spam accounts or messages.
8. **Transfer Learning:** Models pre-trained on large text corpora, such as BERT or GPT, can be fine-tuned for spam classification. Transfer learning leverages the knowledge from general language tasks to improve performance on specific tasks.
9. **Explainability Techniques:** Innovative models can incorporate explainability techniques to provide insights into why a particular message is classified as spam. Interpretability is crucial for user trust and model transparency.
10. **Handling Multimodal Data:** In addition to text, some spam detection systems process images or other data types. Combining text and non-text features in a multimodal approach is an innovative approach.

SUBMISSION:-

PROGRAM:-

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import
TfidfVectorizer, ENGLISH_STOP_WORDS
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score,
confusion_matrix, classification_report
import seaborn as sns
import re
```

 # 1. Data Collection - Assuming the dataset is saved as 'AI_Phase5' in the working directory.

 # 2. Loading the Dataset

```
data=pd.read_csv("C:\\Users\\BYAMUNA1\\Downloads
\\archive\\spam.csv", encoding='latin-1')
```

 # 3. Exploratory Data Analysis (EDA)

```
# Distribution of spam vs. non-spam messages
spam_counts = data['v1'].value_counts()
plt.figure(figsize=(10, 5))
sns.barplot(x=spam_counts.index,
y=spam_counts.values)
plt.xlabel('Label')
plt.ylabel('Count')
```

```
plt.title('Distribution of Spam vs. Non-Spam Messages')
plt.show()

# Message Length Analysis
data['message_length'] = data['v2'].apply(len)
sns.histplot(data, x='message_length', hue='v1',
common_norm=False, alpha=0.5)
plt.xlabel('Message Length')
plt.title('Message Length Distribution by Label')
plt.show()

# Scatter Plot for Message Length vs. Label
plt.figure(figsize=(10, 5))
sns.scatterplot(x='message_length', y=data.index,
hue='v1', data=data, alpha=0.5)
plt.xlabel('Message Length')
plt.ylabel('Index')
plt.title('Scatter Plot of Message Length vs. Label')
plt.show()

# Box Plot of Message Length by Label
plt.figure(figsize=(10, 5))
sns.boxplot(x='v1', y='message_length', data=data)
plt.xlabel('Label')
plt.ylabel('Message Length')
plt.title('Box Plot of Message Length by Label')
plt.show()

# Most Frequent Words in Spam Messages
spam_messages = data[data['v1'] == 'spam']['v2']
spam_words = ' '.join(spam_messages).lower()
```

```
spam_words = re.sub(r'[^a-zA-Z\s]', '', spam_words)
spam_words = ' '.join([word for word in
spam_words.split() if word not in
ENGLISH_STOP_WORDS])
spam_word_counts =
pd.Series(spam_words.split()).value_counts().head(10)

plt.figure(figsize=(10, 5))
sns.barplot(x=spam_word_counts.values,
y=spam_word_counts.index, palette='viridis')
plt.xlabel('Count')
plt.ylabel('Word')
plt.title('Top 10 Most Frequent Words in Spam
Messages')
plt.show()

# Most Frequent Words in Non-Spam Messages
ham_messages = data[data['v1'] == 'ham']['v2']
ham_words = ' '.join(ham_messages).lower()
ham_words = re.sub(r'[^a-zA-Z\s]', '', ham_words)
ham_words = ' '.join([word for word in
ham_words.split() if word not in
ENGLISH_STOP_WORDS])
ham_word_counts =
pd.Series(ham_words.split()).value_counts().head(10)

plt.figure(figsize=(10, 5))
sns.barplot(x=ham_word_counts.values,
y=ham_word_counts.index, palette='viridis')
plt.xlabel('Count')
plt.ylabel('Word')
```

```
plt.title('Top 10 Most Frequent Words in Non-Spam
Messages')
plt.show()

# 4. Data Preprocessing
# Remove duplicates and lowercase the text
data = data.drop_duplicates(keep='first')
data['v2'] = data['v2'].str.lower()

# 5. Text Cleaning and Preprocessing
def clean_text(text):
    # Remove special characters, numbers, and
punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return text

data['v2'] = data['v2'].apply(clean_text)

# 6. Basic Data Processing Methods
print("\nBasic Data Processing Methods:")
print("First 5 rows of the dataset:\n")
print(data.head())
print("\nLast 5 rows of the dataset:\n")
print(data.tail())
print("\nShape of the dataset:", data.shape)
print("\nSize of the dataset:", data.size)
print("\nInfo on missing values (NaN):\n",
data.isna().sum())
print("\nDescriptive statistics:\n", data.describe())
print("\nNumber of unique values in 'v1' (Label):",
data['v1'].nunique())
```

```
print("\nValue counts of 'v1' (Label):\n",
data['v1'].value_counts())
print("\nColumns in the dataset:", data.columns)

# Save the preprocessed dataset to a CSV file
data.to_csv('preprocessed_sms_data.csv', index=False)

# 7. Data Splitting - Split the dataset into training and
testing sets
X_train, X_test, y_train, y_test =
train_test_split(data['v2'], data['v1'], test_size=0.2,
random_state=42)
print("\nShapes of the training and testing sets")
print("\nTraining set shape:", X_train.shape,
y_train.shape)
print("\nTesting set shape:", X_test.shape, y_test.shape)

# Load the preprocessed data
data = pd.read_csv('preprocessed_sms_data.csv',
encoding='latin-1')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(data['v2'], data['v1'], test_size=0.2,
random_state=42)

# Print Training and Testing Set Shapes
print(f"Training set shape: {X_train.shape},
{y_train.shape}")
print(f"Testing set shape: {X_test.shape},
{y_test.shape}")
```

```
# Create a TF-IDF vectorizer to convert text data to numerical features
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Print TF-IDF Vectorizer Information
print(f"TF-IDF Vectorizer Information:")
print(f"Vocabulary size: {len(tfidf_vectorizer.vocabulary_)}")
print(f"Number of terms: {X_train_tfidf.shape[1]}")

# Choose a machine learning algorithm (e.g., Multinomial Naive Bayes)
model = MultinomialNB()

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

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

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label='spam')
recall = recall_score(y_test, y_pred, pos_label='spam')
f1 = f1_score(y_test, y_pred, pos_label='spam')
```

```
# Print Evaluation Metrics
print("\nEvaluation Metrics:")
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

# Cross-Validation
cv_scores = cross_val_score(model, X_train_tfidf,
y_train, cv=5)
confusion = confusion_matrix(y_test, y_pred)  # Add this
line
class_report = classification_report(y_test, y_pred)

print("\nCross-Validation Scores:")
print(cv_scores)
print("Mean CV Score:", cv_scores.mean())

print("\nConfusion Matrix:")
print(confusion)

print("\nClassification Report:")
print(class_report)
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
import seaborn as sns
import re

# 1. Data Collection - Assuming the dataset is saved as 'AI_Phase5' in the working directory.

# 2. Loading the Dataset
data = pd.read_csv("C:\\Users\\BYAMUNA1\\Downloads\\archive\\spam.csv", encoding='latin-1')

# 3. Exploratory Data Analysis (EDA)
# Distribution of spam vs. non-spam messages
spam_counts = data['v1'].value_counts()
plt.figure(figsize=(10, 5))
sns.barplot(x=spam_counts.index, y=spam_counts.values)
plt.xlabel('Label')
plt.ylabel('Count')
plt.title('Distribution of Spam vs. Non-Spam Messages')
plt.show()

# Message Length Analysis
data['message_length'] = data['v2'].apply(len)
sns.histplot(data, x='message_length', hue='v1', common_norm=False, alpha=0.5)
plt.xlabel('Message Length')
plt.title('Message Length Distribution by Label')
plt.show()

# Scatter Plot for Message Length vs. Label
plt.figure(figsize=(10, 5))
sns.scatterplot(x='message_length', y=data.index, hue='v1', data=data, alpha=0.5)
plt.xlabel('Message Length')
plt.ylabel('Index')
plt.title('Scatter Plot of Message Length vs. Label')
plt.show()
```

```
# Box Plot of Message Length by Label
plt.figure(figsize=(10, 5))
sns.boxplot(x='v1', y='message_length', data=data)
plt.xlabel('Label')
plt.ylabel('Message Length')
plt.title('Box Plot of Message Length by Label')
plt.show()

# Most Frequent Words in Spam Messages
spam_messages = data[data['v1'] == 'spam']['v2']
spam_words = ' '.join(spam_messages).lower()
spam_words = re.sub(r'[^a-zA-Z\s]', '', spam_words)
spam_words = ' '.join([word for word in spam_words.split() if word not in ENGLISH_STOP_WORDS])
spam_word_counts = pd.Series(spam_words.split()).value_counts().head(10)

plt.figure(figsize=(10, 5))
sns.barplot(x=spam_word_counts.values, y=spam_word_counts.index, palette='viridis')
plt.xlabel('Count')
plt.ylabel('Word')
plt.title('Top 10 Most Frequent Words in Spam Messages')
plt.show()

# Most Frequent Words in Non-Spam Messages
ham_messages = data[data['v1'] == 'ham']['v2']
ham_words = ' '.join(ham_messages).lower()
ham_words = re.sub(r'[^a-zA-Z\s]', '', ham_words)
ham_words = ' '.join([word for word in ham_words.split() if word not in ENGLISH_STOP_WORDS])
ham_word_counts = pd.Series(ham_words.split()).value_counts().head(10)

plt.figure(figsize=(10, 5))
sns.barplot(x=ham_word_counts.values, y=ham_word_counts.index, palette='viridis')
plt.xlabel('Count')
plt.ylabel('Word')
plt.title('Top 10 Most Frequent Words in Non-Spam Messages')
plt.show()
```

```
# 4. Data Preprocessing
# Remove duplicates and lowercase the text
data = data.drop_duplicates(keep='first')
data['v2'] = data['v2'].str.lower()

# 5. Text Cleaning and Preprocessing
def clean_text(text):
    # Remove special characters, numbers, and punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return text

data['v2'] = data['v2'].apply(clean_text)

# 6. Basic Data Processing Methods
print("\nBasic Data Processing Methods:")
print("First 5 rows of the dataset:\n")
print(data.head())
print("\nLast 5 rows of the dataset:\n")
print(data.tail())
print("\nShape of the dataset:", data.shape)
print("\nSize of the dataset:", data.size)
print("\nInfo on missing values (NaN):\n", data.isna().sum())
print("\nDescriptive statistics:\n", data.describe())
print("\nNumber of unique values in 'v1' (Label):", data['v1'].nunique())
print("\nValue counts of 'v1' (Label):\n", data['v1'].value_counts())
print("\nColumns in the dataset:", data.columns)

# Save the preprocessed dataset to a CSV file
data.to_csv('preprocessed_sms_data.csv', index=False)

# 7. Data Splitting - Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data['v2'], data['v1'], test_size=0.2, random_state=42)
print("\nShapes of the training and testing sets")
print("\nTraining set shape:", X_train.shape, y_train.shape)
print("\nTesting set shape:", X_test.shape, y_test.shape)

# Load the preprocessed data
data = pd.read_csv('preprocessed_sms_data.csv', encoding='latin-1')
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data['v2'], data['v1'], test_size=0.2, random_state=42)

# Print Training and Testing Set Shapes
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Testing set shape: {X_test.shape}, {y_test.shape}")

# Create a TF-IDF vectorizer to convert text data to numerical features
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Print TF-IDF Vectorizer Information
print(f"TF-IDF Vectorizer Information:")
print(f"Vocabulary size: {len(tfidf_vectorizer.vocabulary_)}")
print(f"Number of terms: {X_train_tfidf.shape[1]}")

# Choose a machine learning algorithm (e.g., Multinomial Naive Bayes)
model = MultinomialNB()

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

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

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label='spam')
recall = recall_score(y_test, y_pred, pos_label='spam')
f1 = f1_score(y_test, y_pred, pos_label='spam')

# Print Evaluation Metrics
print("\nEvaluation Metrics:")
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
```

```
# Cross-Validation
cv_scores = cross_val_score(model, X_train_tfidf, y_train, cv=5)
confusion = confusion_matrix(y_test, y_pred)  # Add this line
class_report = classification_report(y_test, y_pred)

print("\nCross-Validation Scores:")
print(cv_scores)
print("Mean CV Score:", cv_scores.mean())

print("\nConfusion Matrix:")
print(confusion)

print("\nClassification Report:")
print(class_report)
```

```
# Print Classification Report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
```

OUTPUT:-

Basic Data Processing Methods:
First 5 rows of the dataset:

```
    v1  ... Unnamed: 4
0   ham  ...       NaN
1   ham  ...       NaN
2  spam  ...       NaN
3   ham  ...       NaN
4   ham  ...       NaN

[5 rows x 5 columns]
```

Last 5 rows of the dataset:

```
      v1  ... Unnamed: 4
5567  spam  ...       NaN
5568   ham  ...       NaN
5569   ham  ...       NaN
5570   ham  ...       NaN
5571   ham  ...       NaN

[5 rows x 5 columns]
```

Shape of the dataset: (5169, 5)

Size of the dataset: 25845

```
Info on missing values (NaN):
 v1              0
v2              0
Unnamed: 2    5126
Unnamed: 3    5159
Unnamed: 4    5164
dtype: int64

Descriptive statistics:
         v1    v2  ...           Unnamed: 3
Unnamed: 4
count   5169  5169  ...                10                         5
unique     2  5116  ...                10                         5
top      ham    ok  ...   MK17 92H. 450Ppw 16"   just Keep-
in-touch\" gdeve.."
freq    4516     4  ...                 1                         1

[4 rows x 5 columns]

Number of unique values in 'v1' (Label): 2

Value counts of 'v1' (Label):
 v1
ham     4516
spam     653
Name: count, dtype: int64

Columns in the dataset: Index(['v1', 'v2', 'Unnamed: 2',
'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

Shapes of the training and testing sets

Training set shape: (4135,) (4135,)

Testing set shape: (1034,) (1034,)

TF-IDF Vectorizer Information:
Vocabulary size: 7547
Number of terms: 7547

Evaluation Metrics:
Accuracy: 0.946085106382979
Precision: 1.0
Recall: 0.586466165413533 8
F1 Score: 0.7393649289099 52

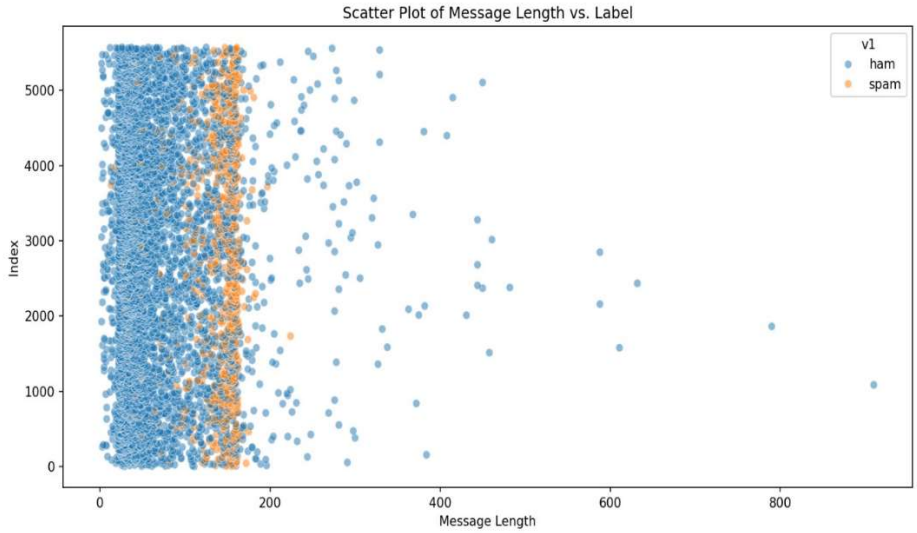
Cross-Validation Scores:
[0.94558646 0.9407497 0.94195889 0.93712213
0.94067797]
Mean CV Score: 0.9412190273194925

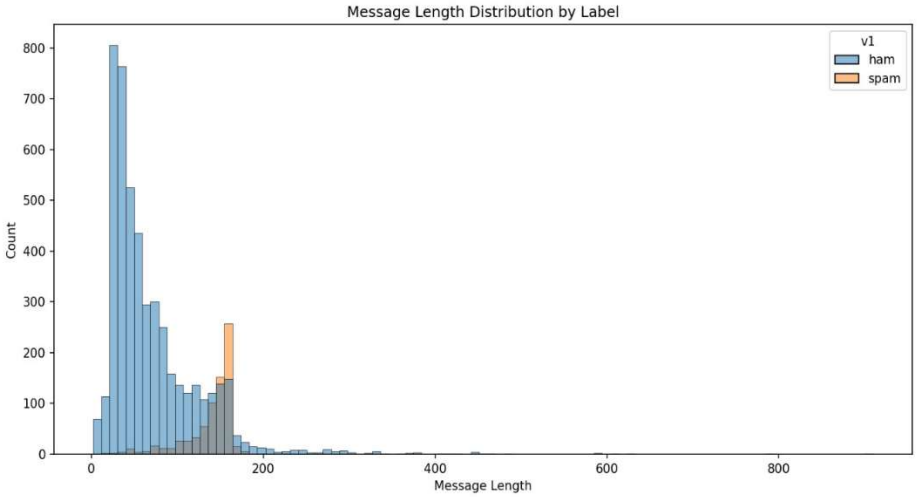
Confusion Matrix:
[[901 0]
[55 78]]

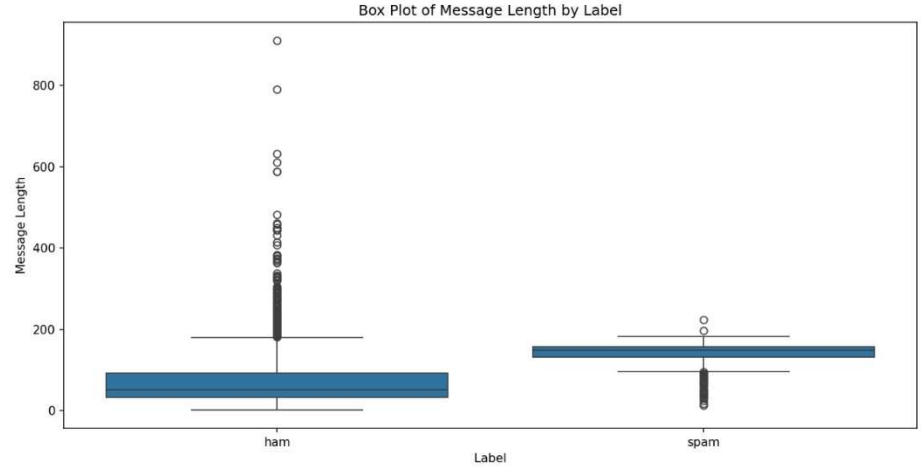
Classification Report:

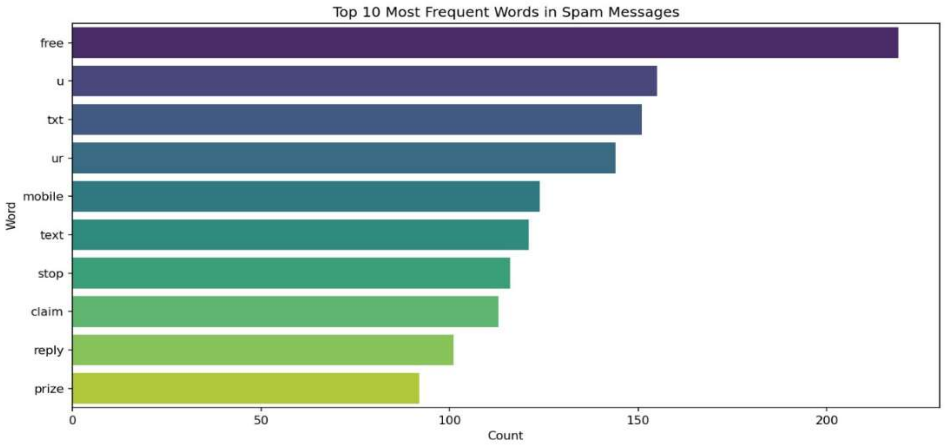
	precision	recall	f1-score	support
ham	0.94	1.00	0.97	901
spam	1.00	0.59	0.74	133
accuracy			0.95	1034
macro avg	0.97	0.79	0.85	1034
weighted avg	0.95	0.95	0.94	1034

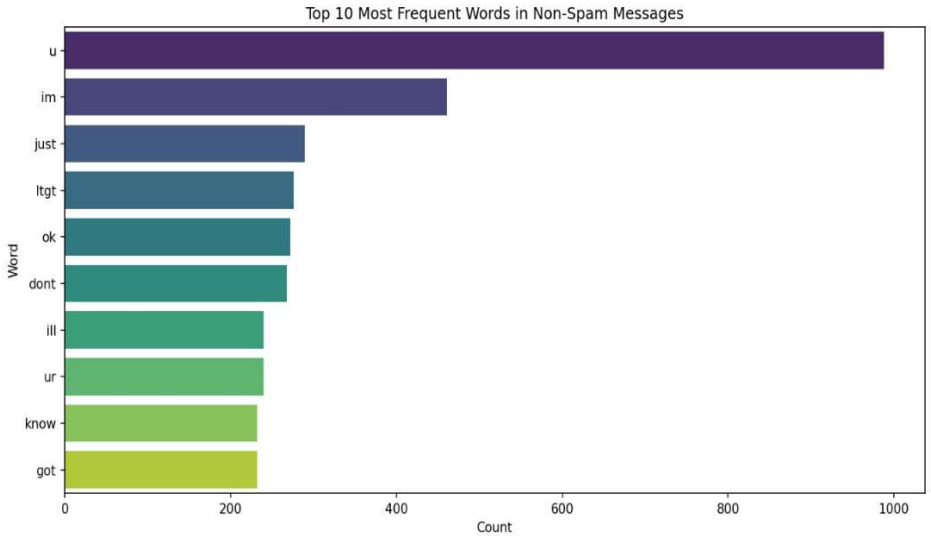
```
Basic Data Processing Methods:
First 5 rows of the dataset:

     v1  ... Unnamed: 4
0   ham  ...        NaN
1   ham  ...        NaN
2  spam  ...        NaN
3   ham  ...        NaN
4   ham  ...        NaN

[5 rows x 5 columns]

Last 5 rows of the dataset:

        v1  ... Unnamed: 4
5567  spam  ...        NaN
5568   ham  ...        NaN
5569   ham  ...        NaN
5570   ham  ...        NaN
5571   ham  ...        NaN

[5 rows x 5 columns]

Shape of the dataset: (5169, 5)

Size of the dataset: 25845

Info on missing values (NaN):
 v1              0
v2               0
Unnamed: 2    5126
Unnamed: 3    5159
Unnamed: 4    5164
dtype: int64

Descriptive statistics:
          v1  ...                          Unnamed: 4
count   5169  ...                                   5
unique     2  ...                                   5
top      ham  ...   just Keep-in-touch\" gdeve.."
freq    4516  ...                                   1

[5 rows x 5 columns]

Shape of the dataset: (5169, 5)

Size of the dataset: 25845

Info on missing values (NaN):
 v1              0
v2               0
Unnamed: 2    5126
Unnamed: 3    5159
Unnamed: 4    5164
dtype: int64

Descriptive statistics:
          v1  ...                          Unnamed: 4
count   5169  ...                                   5
unique     2  ...                                   5
top      ham  ...   just Keep-in-touch\" gdeve.."
freq    4516  ...                                   1

[4 rows x 5 columns]

Number of unique values in 'v1' (Label): 2

Value counts of 'v1' (Label):
 v1
ham     4516
spam     653
Name: count, dtype: int64

Columns in the dataset: Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')

Shapes of the training and testing sets


Training set shape: (4135,) (4135,)

Testing set shape: (1034,) (1034,)
TF-IDF Vectorizer Information:
Vocabulary size: 7673
Number of terms: 7673

Evaluation Metrics:
Accuracy: 0.9545454545454546
Precision: 1.0
Recall: 0.6758620689655173
F1 Score: 0.8065843621399177

Cross-Validation Scores:
[0.95525998 0.94800484 0.95042322 0.94921403 0.93954051]
Mean CV Score: 0.9484888126964933

Confusion Matrix:
[[889   0]
 [ 47  98]]

Classification Report:
              precision    recall  f1-score   support

         ham       0.95      1.00      0.97       889
        spam       1.00      0.68      0.81       145

    accuracy                           0.95      1034
   macro avg       0.97      0.84      0.89      1034
weighted avg       0.96      0.95      0.95      1034
```

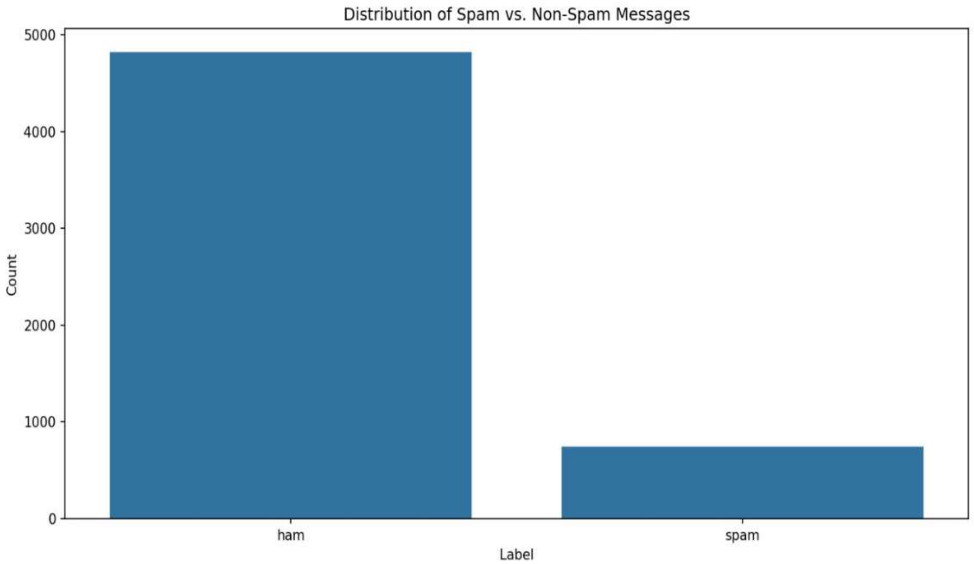

Scatter Plot of Message Length vs. Label
v1
ham
spam
Index
Message Length
0
1000
2000
3000
4000
5000
0
200
400
600
800


Message Length Distribution by Label
v1
ham
spam
Count
Message Length
0
100
200
300
400
500
600
700
800
0
200
400
600
800


Box Plot of Message Length by Label
Message Length
Label
ham
spam
0
200
400
600
800


Top 10 Most Frequent Words in Spam Messages
free
u
txt
ur
mobile
text
stop
claim
reply
prize
Word
0
50
100
150
200
Count


Top 10 Most Frequent Words in Non-Spam Messages
u
im
just
ltgt
ok
dont
ill
ur
know
got
Word
0
200
400
600
800
1000
Count


Distribution of Spam vs. Non-Spam Messages
5000
4000
3000
2000
1000
0
Count
ham
spam
Label