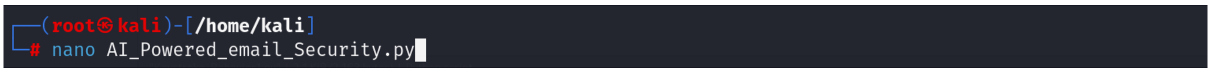
AI Powered email Security

Email has become an essential communication tool in today's digital age. However, with the increasing sophistication of cyber threats, ensuring email security has never been more critical. This project delves into an AI-powered Email Security Application that combines two powerful machine learning models: one for detecting spam or phishing emails, and another for identifying malicious URLs. By the end of this guide, you'll have a comprehensive understanding of how this application works, from data preprocessing to model training and prediction.

To begin writing the code open the nano editor using the command:

```
$ nano AI_Powered_email_Security.py
```

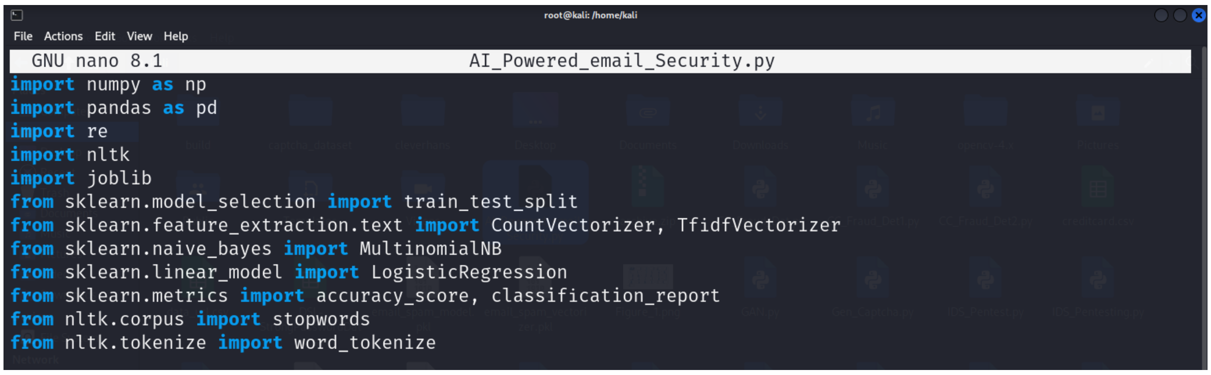


```
┌──(root㉿kali)-[/home/kali]
└─# nano AI_Powered_email_Security.py
```

The entire code is available at https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Capstone%20Projects/AI_Powered_email_Security.py

Also download the 2 required datasets from https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Capstone%20Projects/spam.csv and https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Capstone%20Projects/data_url.csv

1. Importing Necessary Libraries



```
import numpy as np
import pandas as pd
import re
import nltk
import joblib
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

numpy and pandas are used for data manipulation and analysis.

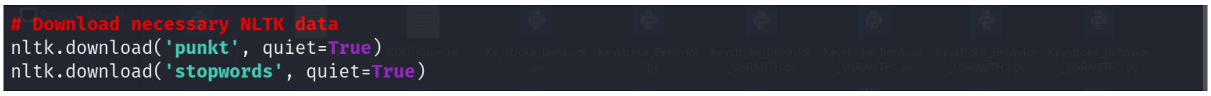
re provides support for regular expressions.

nltk (Natural Language Toolkit) is used for natural language processing tasks.

joblib is used for saving and loading machine learning models.

Various modules from sklearn (Scikit-learn) are imported for machine learning tasks such as data splitting, feature extraction, model training, and evaluation.

2. Downloading NLTK Data

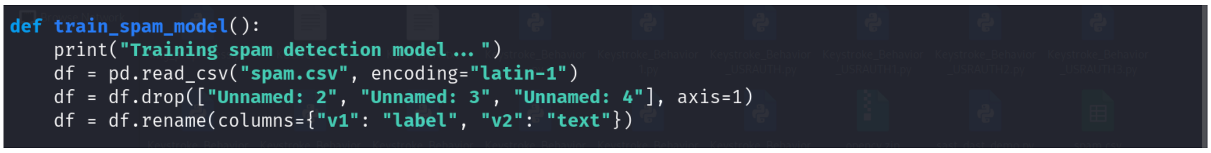


```
# Download necessary NLTK data
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
```

These lines download necessary NLTK data for tokenization (punkt) and a list of common words to ignore (stopwords). The quiet=True parameter suppresses the download progress messages.

3. Spam Detection Model

3.1 Training the Spam Detection Model



```
def train_spam_model():
    print("Training spam detection model ... ")
    df = pd.read_csv("spam.csv", encoding="latin-1")
    df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
    df = df.rename(columns={"v1": "label", "v2": "text"})
```

This function starts by loading the spam dataset from a CSV file. It then drops unnecessary columns and renames the remaining columns for clarity.

```
# Preprocess text data
df["text"] = df["text"].str.lower()
df["text"] = df["text"].apply(word_tokenize)
stop_words = set(stopwords.words("english"))
df["text"] = df["text"].apply(lambda x: [word for word in x if word not in stop_words])
df["text"] = df["text"].apply(lambda x: " ".join(x))
```

These lines preprocess the text data:

1. Convert all text to lowercase.
2. Tokenize the text (split it into individual words).
3. Remove stop words (common words like "the", "a", "an" that don't carry much meaning).
4. Join the remaining words back into a single string.

```
# Feature Extraction
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df["text"])
```

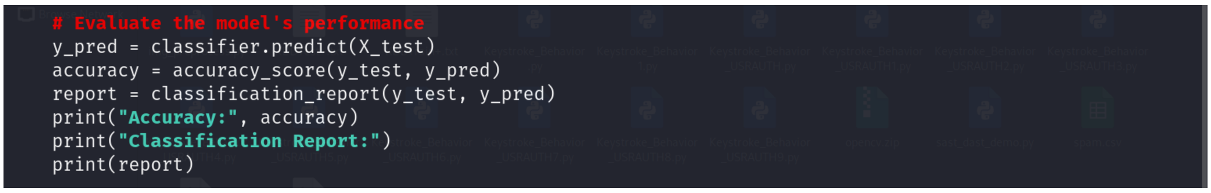
Here, we use CountVectorizer to convert the text data into a matrix of token counts. This is a crucial step in preparing the text data for machine learning.

```
# Split the Dataset
X_train, X_test, y_train, y_test = train_test_split(X, df["label"], test_size=0.2, random_state=42)
```

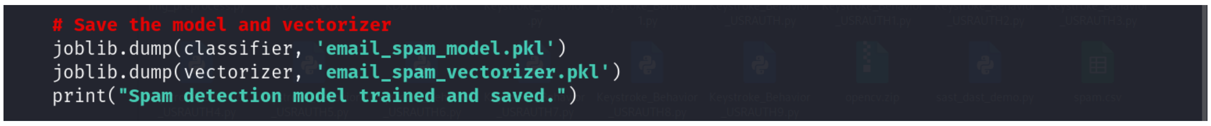
This line splits the data into training and testing sets. 80% of the data is used for training, and 20% for testing.

```
# Train a classification model
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

Here, we create a Multinomial Naive Bayes classifier and train it on the training data.

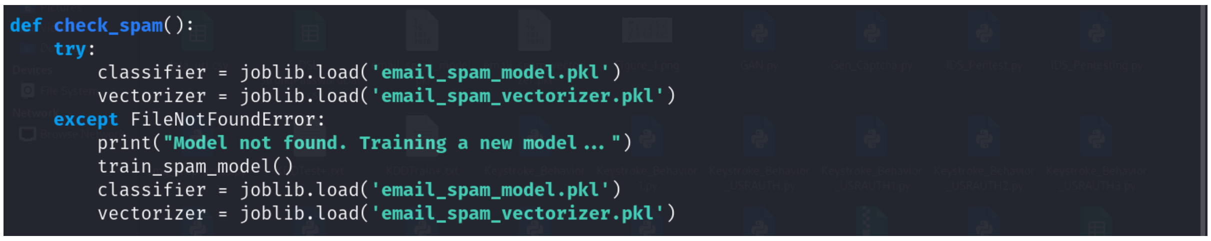

```
# Evaluate the model's performance
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
```

These lines evaluate the model's performance on the test set and print the accuracy and a detailed classification report.

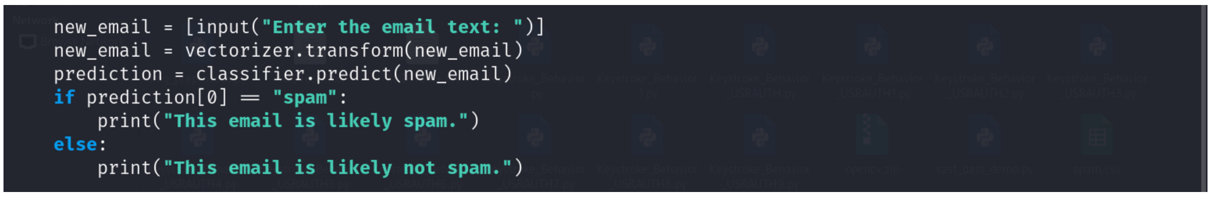

```
# Save the model and vectorizer
joblib.dump(classifier, 'email_spam_model.pkl')
joblib.dump(vectorizer, 'email_spam_vectorizer.pkl')
print("Spam detection model trained and saved.")
```

Finally, we save the trained model and vectorizer to files for later use.

3.2 Checking for Spam


```
def check_spam():
    try:
        classifier = joblib.load('email_spam_model.pkl')
        vectorizer = joblib.load('email_spam_vectorizer.pkl')
    except FileNotFoundError:
        print("Model not found. Training a new model...")
        train_spam_model()
        classifier = joblib.load('email_spam_model.pkl')
        vectorizer = joblib.load('email_spam_vectorizer.pkl')
```

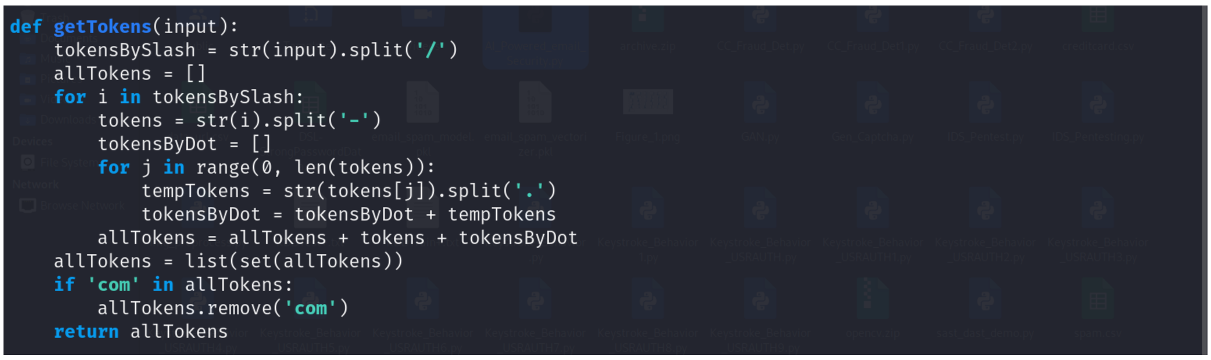
This function attempts to load the saved spam detection model and vectorizer. If they're not found, it trains a new model.


```
    new_email = [input("Enter the email text: ")]
    new_email = vectorizer.transform(new_email)
    prediction = classifier.predict(new_email)
    if prediction[0] == "spam":
        print("This email is likely spam.")
    else:
        print("This email is likely not spam.")
```

These lines take user input, transform it using the loaded vectorizer, and then use the classifier to predict whether the input is spam or not.

4. URL Detection Model

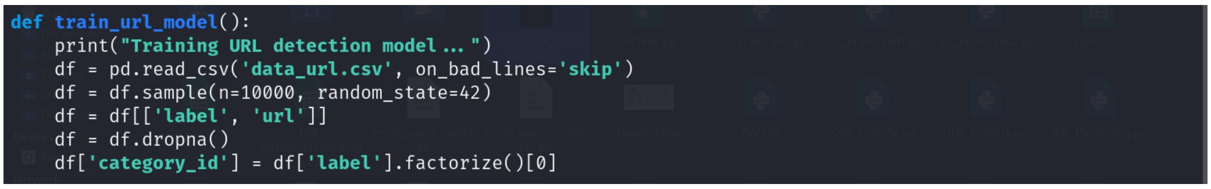
4.1 URL Tokenization



```python
def getTokens(input):
    tokensBySlash = str(input).split('/')
    allTokens = []
    for i in tokensBySlash:
        tokens = str(i).split('-')
        tokensByDot = []
        for j in range(0, len(tokens)):
            tempTokens = str(tokens[j]).split('.')
            tokensByDot = tokensByDot + tempTokens
        allTokens = allTokens + tokens + tokensByDot
    allTokens = list(set(allTokens))
    if 'com' in allTokens:
        allTokens.remove('com')
    return allTokens
```

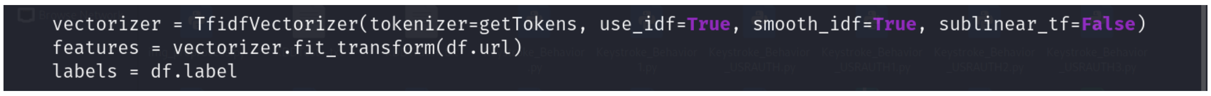
This function tokenizes a URL by splitting it on slashes, hyphens, and dots. It then removes duplicates and the common 'com' token. This custom tokenization is crucial for extracting meaningful features from URLs.

4.2 Training the URL Detection Model



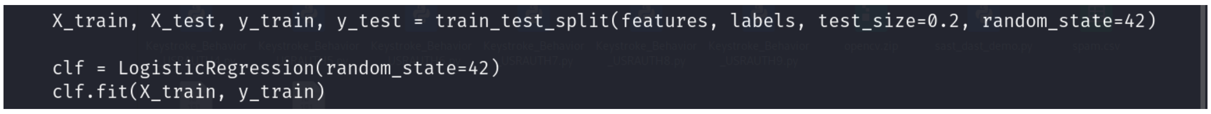
```python
def train_url_model():
    print("Training URL detection model ... ")
    df = pd.read_csv('data_url.csv', on_bad_lines='skip')
    df = df.sample(n=10000, random_state=42)
    df = df[['label', 'url']]
    df = df.dropna()
    df['category_id'] = df['label'].factorize()[0]
```

This function starts by loading the URL dataset, sampling 10,000 entries for faster processing, selecting relevant columns, removing any rows with missing values, and creating a numeric category ID for each label.



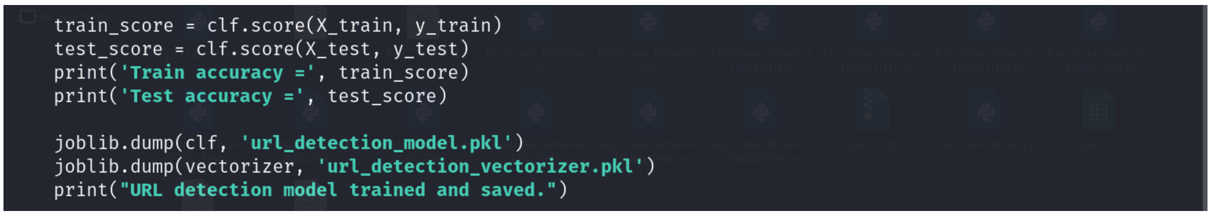
```python
    vectorizer = TfidfVectorizer(tokenizer=getTokens, use_idf=True, smooth_idf=True, sublinear_tf=False)
    features = vectorizer.fit_transform(df.url)
    labels = df.label
```

Here, we use TfidfVectorizer with our custom tokenizer to convert URLs into a matrix of TF-IDF features. TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that reflects how important a word is to a document in a collection.



```python
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

    clf = LogisticRegression(random_state=42)
    clf.fit(X_train, y_train)
```

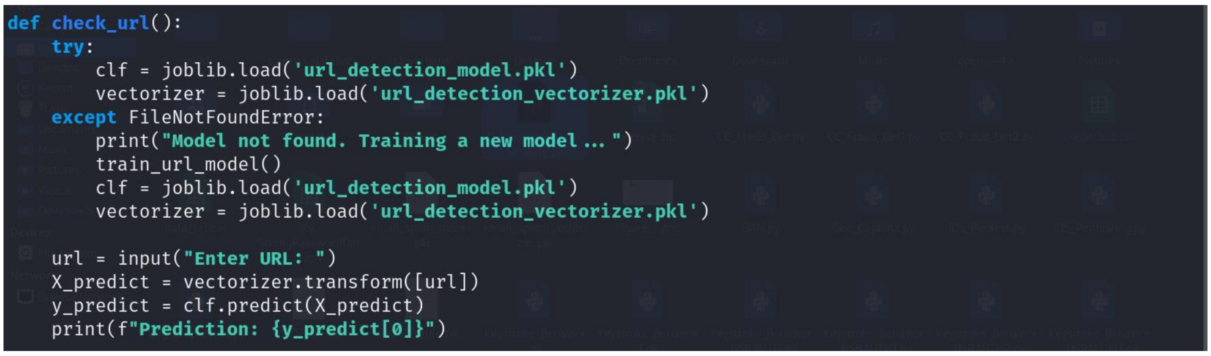
We split the data into training and testing sets, then create and train a Logistic Regression classifier.



```python
    train_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    print('Train accuracy =', train_score)
    print('Test accuracy =', test_score)

    joblib.dump(clf, 'url_detection_model.pkl')
    joblib.dump(vectorizer, 'url_detection_vectorizer.pkl')
    print("URL detection model trained and saved.")
```

These lines evaluate the model's performance on both training and test sets, then save the model and vectorizer to files.

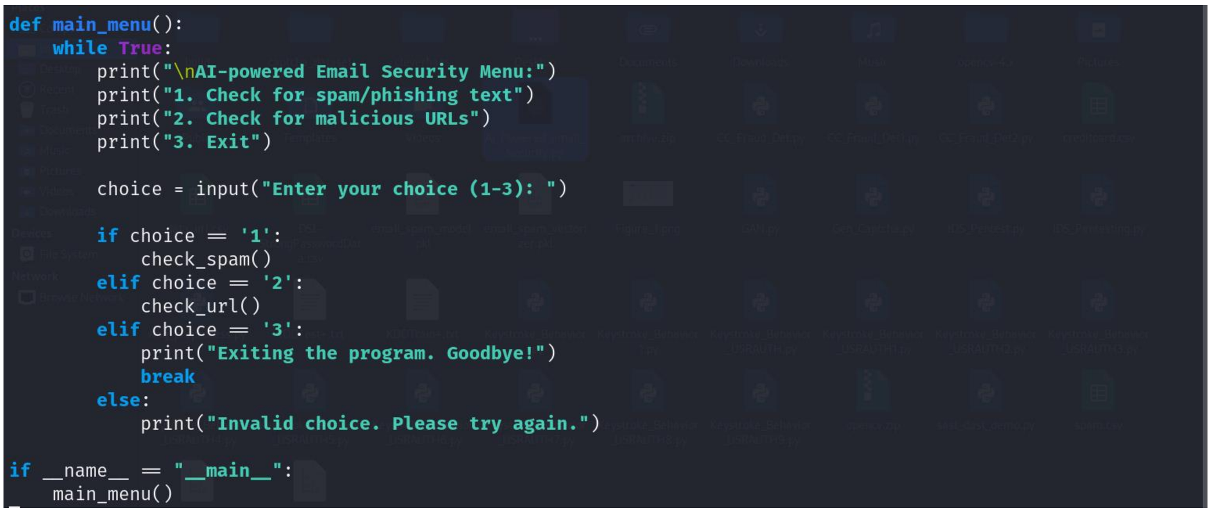
4.3 Checking URLs



```python
def check_url():
    try:
        clf = joblib.load('url_detection_model.pkl')
        vectorizer = joblib.load('url_detection_vectorizer.pkl')
    except FileNotFoundError:
        print("Model not found. Training a new model...")
        train_url_model()
        clf = joblib.load('url_detection_model.pkl')
        vectorizer = joblib.load('url_detection_vectorizer.pkl')

    url = input("Enter URL: ")
    X_predict = vectorizer.transform([url])
    y_predict = clf.predict(X_predict)
    print(f"Prediction: {y_predict[0]}")
```

This function loads the URL detection model and vectorizer (or trains them if not found), takes a URL input from the user, transforms it, and predicts whether it's malicious.

5. Main Menu



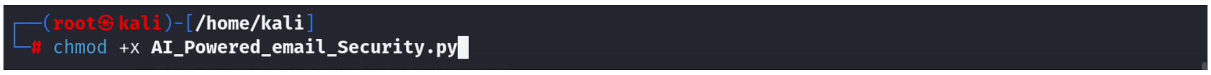
```
def main_menu():
    while True:
        print("\nAI-powered Email Security Menu:")
        print("1. Check for spam/phishing text")
        print("2. Check for malicious URLs")
        print("3. Exit")

        choice = input("Enter your choice (1-3): ")

        if choice == '1':
            check_spam()
        elif choice == '2':
            check_url()
        elif choice == '3':
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main_menu()
```

The main_menu function provides an interactive interface for the user to choose between checking for spam text or malicious URLs. It runs in a loop until the user chooses to exit.

Save the file and close it. (CTRL+x -> y -> Enter)

Convert the file to an executable one using:

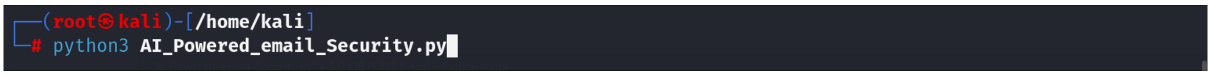
$ `chmod +x AI_Powered_email_Security.py`



```
┌──(root㉿kali)-[/home/kali]
└─# chmod +x AI_Powered_email_Security.py
```

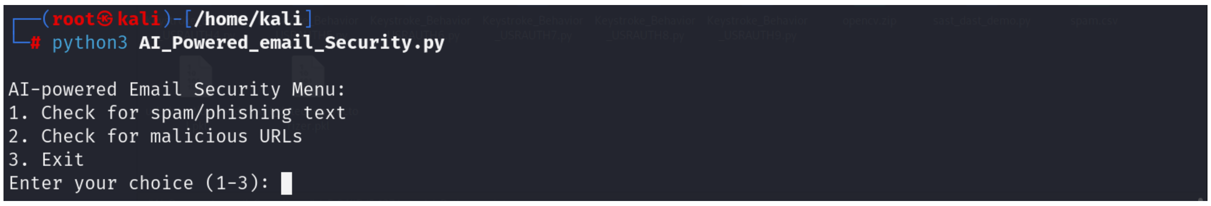
Finally run the program using the command:

$ `python3 AI_Powered_email_Security.py`



```
┌──(root㉿kali)-[/home/kali]
└─# python3 AI_Powered_email_Security.py
```

You can explore the output through the interactive menu.



```
┌──(root㉿kali)-[/home/kali]
└─# python3 AI_Powered_email_Security.py

AI-powered Email Security Menu:
1. Check for spam/phishing text
2. Check for malicious URLs
3. Exit
Enter your choice (1-3):
```

This AI-powered Email Security Application demonstrates the power of machine learning in cybersecurity. By combining natural language processing techniques with advanced classification algorithms, it provides a robust tool for detecting both spam emails and malicious URLs.

The application showcases several key concepts in machine learning and data processing:

1. Data Preprocessing: Both models involve significant data cleaning and preparation steps, crucial for effective machine learning.
2. Feature Extraction: The application uses different techniques (CountVectorizer and TfidfVectorizer) to convert text data into numerical features that machine learning models can understand.
3. Model Training: It demonstrates the use of two different classification algorithms (Naive Bayes and Logistic Regression) for different tasks.
4. Model Persistence: By saving and loading models, the application can reuse trained models without needing to retrain them every time.
5. User Interaction: The menu-driven interface makes the application accessible and easy to use.

While this application provides a strong foundation for email security, it's important to note that in real-world scenarios, these models would need to be regularly updated with new data to stay effective against evolving threats. Additionally, more advanced techniques like deep learning could potentially improve the accuracy of predictions.

By understanding this code, you've taken a significant step into the world of AI-powered cybersecurity. As you continue to learn and experiment, you'll discover even more ways that machine learning can be applied to keep our digital communications safe and secure.