

Name:

Luke Seavey

Date Created:

09/21/25

Program Description:

This program scans an email's text to check the likelihood of it being spammail based on common words found in said emails.

Functions used in the Program (list in order as they are called):

1. Function Name: main

Description:

Main is the CLI entry point that displays the user interface and gathers information and sends it to analyze_email and displays the final values.

Parameters:

- There are no parameters in main

Variables:

- Email_message - is the pasted text from the email the user wants scanned
- Result - is the final calculation on how likely the email is to being scammail
- Item, count, weight - tuple pulled from results

Logical Steps:

1. Asks the user for the email to be scanned
2. Sends text to analyze_email
3. Prints the first half of the user interface
4. Makes a list of words with the flagged words and phrases from the scam email and prints them out
5. Then prints the last half of the user interface.

Returns:

No, text is returned as outputs to the terminal

2. Function Name: analyze_email(email_message: str):

Description:

Main handler that scans words and phrases, gives a compute score, labels and returns a dict with all the info needed.

Parameters:

- Email_message - Is the text that you input from the email that you want to scan.

Variables:

- Word_counts - is the count of the number of words in the pasted text
- Phrase_counts - is the number of phrases found in the text
- Score - the counted score of matched words and phrases
- Label - how likely the email is to being spam

Logical Steps:

1. Send the text to each other function with the dict
2. Send the text to scan_text_for_words and saves answer to word_count
3. Sends phrases to check_phrase_count and saves answer to phrase_count
4. Send word_count and phrase_count to Compute_spam_score and saves answer to score
5. Sends score to Likelihood_label and saves answer to score
6. Makes a list of all the triggers that added score

Returns:

- Returns all the variables with the corresponding values and turns it into a dict

3. Function name: scan_text_for_words(text: str, vocab: set[str]):

Description:

Scans the text for words that match the dict for words commonly used in spam emails.

Parameters:

- Text - user imputed Email text
- Vocab - SPAM_WORDS, this is placed in a set for fast if in checks

Variables:

- Counts - creates a Counter object from Counter class
- Tokens - Processed text with special characters removed

Logical Steps:

1. Creates a counter class
2. Takes the email text and sends it to the normalize function
3. For loop that checks if tokens are in the email text (vocab set)
4. Returns the words that were flagged and amount of times it appeared returns as a counter object

Returns:

Counter object that contains the words that matched the flagged words and the amount of times they were called

3. Function Name: tokenize(text: str):

Description:

Runs text through the normalize function and splits the words up from whitespace

Parameters:

- Text: str - user imputed email text

Variables:

Logical Steps:

1. Sends text to normalize
2. removes white text at the beginning and tokenizes words with .split() and returns it

Returns:

Normalized and tokenized text

4. Function Name: `normalize(text: str):`

Description:

Normalizes text for ease of processing by making text lower case and removing punctuation, emojis and special characters.

Parameters:

- Text: str - user imputed email text

Variables:

- Text - edited text throughout the function

Logical Steps:

1. Takes user entered email text and makes it lower case
2. Replaces any non-alphanumeric characters with spaces
3. Replaces any white space that is one or more with a single whitespace to string and removes white text at the beginning and end of the string and returns it.

Returns:

A the email text without extra whitespace, emojis, and special characters

5. Function Name: `check_phrase_conts(text: str, phrases: set[str]):`

Description:

Scans text for flagged multi-word spam phrases

Parameters:

- Text: str - User provided email text
- Phrases: set[str] - Set of common spam phrases (SPAM_PHRASES)

Variables:

- Counts - Counter of phrase matches
- Padded - Normalizes text surrounded by spaces for boundary checking
- Pattern - Regex pattern built for each phrase
- Hits - list of regex matches found

Logical Steps:

1. Normalize the text and pad it with spaces
2. Build a regex pattern for each phrase
3. Use `re.findall` to detect matches
4. Record matches in counts

Returns:

Counter object containing flagged phrases and their counts

5. Function Name: `compute_spam_score(word_count: counter, phrase_counts: counter):`

Description:

Combines the weighted counts to produce the complete spam score between words and phrases.

Parameters:

- `Word_counts` - Counter of spam words
- `Phrase_counts` - counter of spam phrases

Variables:

- `Word_points` - Total points from words (`count * 1`)
- `phrase_points` - Total points from phrases (`count * 2`)

Logical Steps:

1. Multiply word counts by weight = 1
2. Multiply phrase counts by weight = 2
3. Add them for the total score

Returns:

Integer spam score

5. Function Name: `likelihood_label(score: int):`

Description:

Maps the score and uses it to calculate how likely the email is to be spam and then prints a result text based on ranges that the score could fall in. These can also be changed to be more or less strict.

Parameters:

- Score - Total Spam score

Variables:

- No variables within likelihood_label

Logical Steps:

1. Compare score against thresholds
2. Assign label (low, moderate, high, very high)

Returns:

- Returns the end text to the user summarizing the score to be either low, moderate, high, or very high likelihood to be spammail.

Logical steps:

1. Start program which begins execution of main.
2. The program asks the user to paste the email text into the terminal.
3. Input is saved into the variable email_message.
4. Sends the text to analyze_email

5. main calls analyze_email
6. Inside analyze_email, the email is passed to scan_text_for_words.
7. scan_text_for_words calls tokenize.
8. tokenize calls normalize to format the text.
9. Normalized text is split into tokens (words).
10. Tokens are compared against spam_words set.
11. A Counter of flagged spam words is returned as word_counts.
12. analyze_email also calls check_phrase_counts.
13. Text is normalized and padded.,
14. Each phrase is turned into a regex pattern.
15. Regex finds matches inside the text.
16. A Counter of flagged spam phrases is returned as phrase_counts.
17. Compute spam score
18. analyze_email calls compute_spam_score.
19. Word matches are multiplied by weight = 1.
20. Phrase matches are multiplied by weight = 2.
21. Both totals are added together.
22. Final integer score is returned as score.
23. Assign likelihood label
24. analyze_email calls likelihood_label.
25. Score is checked against thresholds.,
26. Returns a label: Low, Moderate, High, Very High likelihood of spam.
27. Build triggers list
28. analyze_email builds triggers, a list of tuples (item, count, weight) for every flagged word/phrase.
29. The list is sorted by contribution (points) from highest to lowest.
30. Return results.
31. analyze_email returns a dictionary containing:
score,label,word_counts,phrase_counts,triggers
32. Display results to the user.
33. Back in main, the results are printed:
34. Spam score
35. Likelihood label
36. Each flagged word/phrase with details (item, count, weight, points)
37. End program.

List of functions in order:

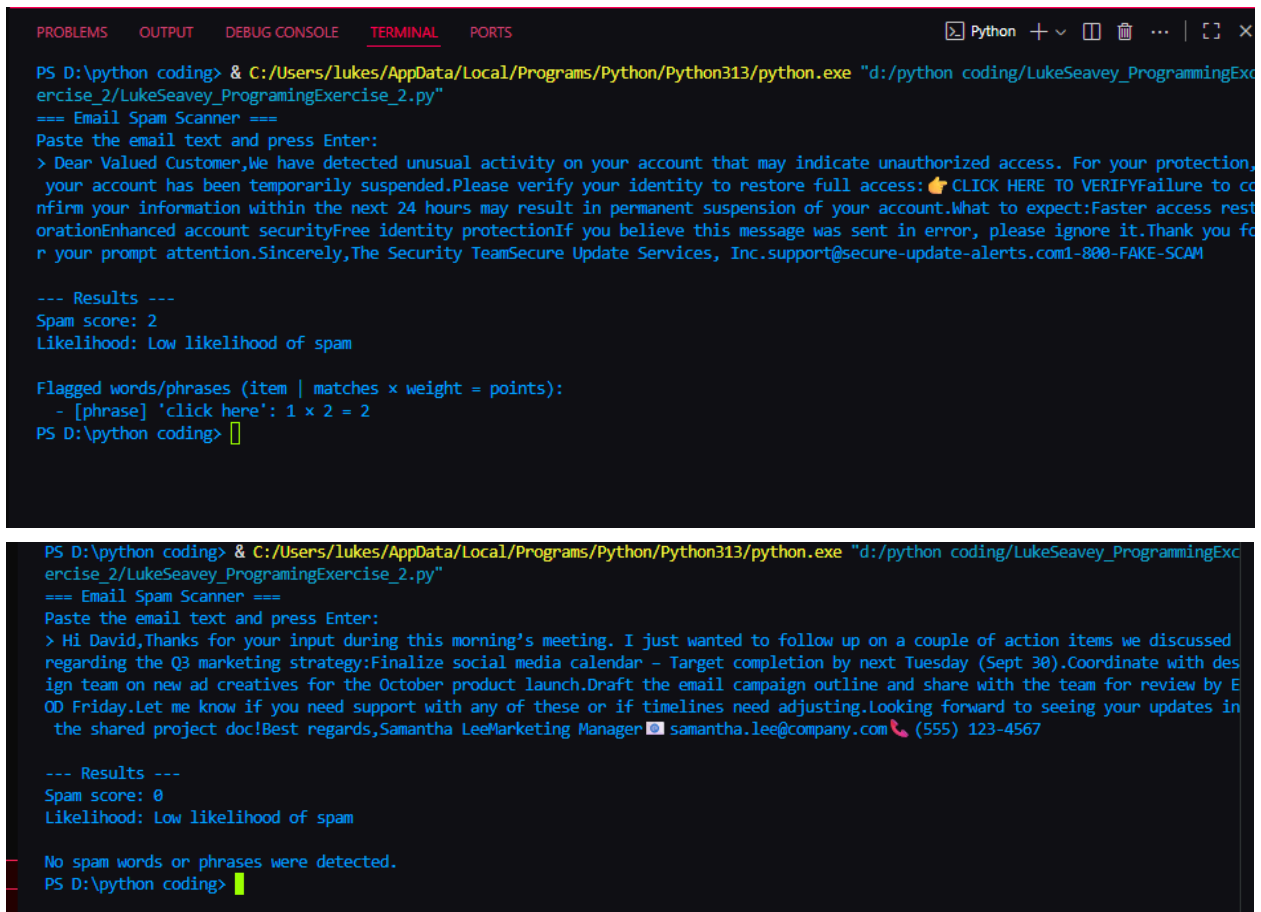
1. Main
2. Analyze_email
3. Scan_text_for_words
4. Tokenize
5. Normalize

6. Check_phrase_count
7. Normalize
8. Compute_spam_score
9. Likelihood_label

Link to your repository:

https://github.com/FennecAce/COP2373/tree/b13d79e774ffb0a62a4ee03ffb8d66bfdfb9e822/LukeSeavey_ProgrammingExercise_2

Output Screenshot:



```
PS D:\python coding> & C:/Users/luke/AppData/Local/Programs/Python/Python313/python.exe "d:/python coding/LukeSeavey_ProgrammingExercise_2/LukeSeavey_ProgrammingExercise_2.py"
=== Email Spam Scanner ===
Paste the email text and press Enter:
> Dear Valued Customer,We have detected unusual activity on your account that may indicate unauthorized access. For your protection, your account has been temporarily suspended.Please verify your identity to restore full access:👉CLICK HERE TO VERIFYFailure to confirm your information within the next 24 hours may result in permanent suspension of your account.What to expect:Faster access restorationEnhanced account securityFree identity protectionIf you believe this message was sent in error, please ignore it.Thank you for your prompt attention.Sincerely,The Security TeamSecure Update Services, Inc.support@secure-update-alerts.com1-800-FAKE-SCAM

--- Results ---
Spam score: 2
Likelihood: Low likelihood of spam

Flagged words/phrases (item | matches x weight = points):
- [phrase] 'click here': 1 x 2 = 2
PS D:\python coding>

PS D:\python coding> & C:/Users/luke/AppData/Local/Programs/Python/Python313/python.exe "d:/python coding/LukeSeavey_ProgrammingExercise_2/LukeSeavey_ProgrammingExercise_2.py"
=== Email Spam Scanner ===
Paste the email text and press Enter:
> Hi David,Thanks for your input during this morning's meeting. I just wanted to follow up on a couple of action items we discussed regarding the Q3 marketing strategy:Finalize social media calendar - Target completion by next Tuesday (Sept 30).Coordinate with design team on new ad creatives for the October product launch.Draft the email campaign outline and share with the team for review by EOD Friday.Let me know if you need support with any of these or if timelines need adjusting.Looking forward to seeing your updates in the shared project doc!Best regards,Samantha LeeMarketing Manager📧 samantha.lee@company.com📞 (555) 123-4567

--- Results ---
Spam score: 0
Likelihood: Low likelihood of spam

No spam words or phrases were detected.
PS D:\python coding>
```