**Output Analysis of**

**Email Slicer Python Project**

By

*Debanik Deb*

**Email Slicer Python Project Output**

The Email Slicer Python project allows users to enter email details and categorize them into different folders such as inbox, spam, important, social, and starred. The program collects the email details, organizes them based on the folder, and saves them to separate CSV files. Finally, it merges all the CSV files into a single DataFrame for further analysis.

**The Output**

The output of the program follows a structured flow where the user is prompted to enter email details for each folder. The user is initially asked to enter the necessary details of the emails, including the folder name, email ID, subject, date, and time. The program provides a loop that allows the user to add multiple emails within a folder. Once all the emails are entered, the user is given the option to enter another folder or exit the program.

The program then proceeds to save the entered emails to separate CSV files based on their respective folders. It creates separate directories for each folder and generates CSV files containing the email details. The user receives a success message indicating that the emails have been saved successfully.

After saving the emails to CSV files, the program merges all the CSV files into a single DataFrame called `merged\_df`. If the merging process is successful, the program displays the head of the merged DataFrame, showing the email details from all folders in a tabular format.

Finally, the program concludes with a message indicating that the email slicing operation is done.

**The Flow of the Program**

1. The user is prompted to enter the necessary details of the emails.

2. The user enters the folder name (inbox, spam, important, social and starred).

3. The user enters the email ID, subject, date, and time.

4. The user has the option to add another email within the same folder.

5. Once all the emails are added to the current folder, the user is prompted to enter another folder or exit the program.

6. The program saves the entered emails to separate CSV files based on their folders.

7. The program merges all the CSV files into a single DataFrame.

8. If the merging process is successful, the program displays the head of the merged DataFrame.

9. The program concludes with a message indicating the completion of the email slicing operation.

**How the Output Works with the Flow of the Program**

* The output of the program is designed to provide a clear and interactive experience for the user. It follows the flow of the program by requesting input from the user at each step and providing relevant feedback and information.
* The program prompts the user to enter the folder name, email ID, subject, date, and time for each email. It validates the user's input and ensures that the entered data is correctly recorded and categorized.
* After the user finishes adding emails to a folder, the program offers the option to enter another folder or exit the program. This allows the user to add emails to multiple folders in a convenient and organized manner.
* Once the user completes entering emails and chooses to exit the program, the program saves the emails to separate CSV files based on their folders. It creates directories for each folder if they don't already exist and generates CSV files containing the email details.
* After saving the emails, the program merges all the CSV files into a single DataFrame. If the merging process is successful, the program displays the head of the merged DataFrame, providing the user with a visual representation of the overall email data.
* The program concludes by notifying the user that the email slicing operation is done, indicating the successful completion of the task.

**The New Dataset**

After merging the CSV files, the resulting DataFrame is `merged\_df`. It contains the following columns: 'EmailID', 'Subject', 'Date', and 'Time'. These columns represent the email details that were entered by the user.

***Output:***

The 'EmailID' column contains the email addresses of the senders, the 'Subject' column contains the subject lines of the emails, the 'Date' column contains the dates when the emails were received, and the 'Time' column contains the timestamps of the emails.

***Visualization of the New Dataset***

To visualize the merged dataset, we can use various plots and charts. Here are two examples:

*1. Count Plot of Dates*

This code generates a count plot that displays the frequency of emails based on the dates they were received. The x-axis represents the dates, and the y-axis represents the count of emails received on each date. The `palette="rainbow"` argument sets the color palette for the plot.

This visualization helps us understand the distribution of emails over time and identify any patterns or trends in email activity.

*2. Count Plot of Email IDs*

This code generates a count plot that displays the frequency of emails based on the email IDs of the senders. The y-axis represents the email IDs, and the x-axis represents the count of emails sent by each sender. The `palette="GnBu"` argument sets the color palette for the plot.

This visualization provides insights into the email activity of different senders and helps identify the most active or frequently encountered email addresses.

Analytically, these visualizations allow us to analyze the distribution of emails across dates and identify any significant trends or patterns. The count plot of dates helps us understand if there are any specific dates with a higher or lower volume of emails, which could be useful for analyzing email traffic. The count plot of email IDs helps identify the most frequent senders or the ones that appear more frequently in the dataset, which can provide insights into the email sources or potential areas of focus.

These visualizations provide a quick overview of the merged email dataset and assist in exploratory data analysis, allowing for further insights into the email data and facilitating decision-making based on the patterns observed.

**Conclusion**

The Email Slicer Python project demonstrates an effective way to collect, organize, and analyze email data. By allowing users to enter email details, categorize them into folders, and save them to CSV files, the program enables efficient data management. The merging of CSV files into a single DataFrame further facilitates data analysis and exploration.

The project is useful for data analysis purposes as it provides a streamlined process for collecting and storing email data in a structured format. The generated CSV files can be easily imported into other tools or libraries for further analysis, such as Pandas or machine learning algorithms.

Overall, the Email Slicer Python project offers a practical solution for organizing and managing email data, making it a valuable tool for data analysts, researchers, and anyone working with email data.