**Email Slicer Python Project**

By

*Debanik Deb*

**Introduction**

The Email Slicer Python Project is designed to provide a solution for managing and organizing emails by slicing them into different folders based on user-defined criteria. The project allows users to enter email details such as email ID, subject, date, and time, and categorize them into folders such as inbox, spam, important, social, or starred. The project utilizes the `EmailManager` class to handle email operations and provides functionality to save the emails in separate CSV files and merge them into a single DataFrame.

**Email Manager Class**

**Description**

The `EmailManager` class acts as a manager for handling email operations, including adding emails, saving emails to CSV files, and grouping emails by folder.

**Functions**

***\_\_init\_\_ (self, data\_path)***

This function initializes the `EmailManager` object with a `data\_path` parameter, which represents the path where the emails will be stored. It assigns the provided `data\_path` to the `data\_path` attribute of the `EmailManager` object. This attribute will be used later when saving the emails to CSV files.

***add\_email (self, email\_id, email\_subject, email\_date, email\_time, folder)***

The `add\_email` function is responsible for adding an email to the manager's email collection. It takes parameters such as `email\_id`, `email\_subject`, `email\_date`, `email\_time`, and `folder`. These parameters represent the details of the email, including the email ID, subject, date, time, and the folder to which it belongs.

Inside the function, the email details are stored in a dictionary format using the provided parameters. The email ID is used as the key, and the corresponding email details are stored as a value in the `emails` attribute of the `EmailManager` object.

***save\_emails (self)***

The `save\_emails` function is responsible for saving the emails to separate CSV files based on their folders. It creates separate folders for each category if they don't already exist and generates CSV files containing the email details.

Inside the function, the `data\_path` attribute is used as the base directory where the email folders and CSV files will be created. The function iterates through the emails grouped by folder using the `group\_emails\_by\_folder` function. For each folder, it creates a folder directory using the `folder\_path` variable and checks if it already exists. If the folder doesn't exist, it creates the folder using the `mkdir` function.

Next, the function generates the file name for the CSV file based on the folder name. It creates the file path using the `folder\_path` and `file\_name` variables. The email details for the current folder are extracted and converted into a Pandas DataFrame. The DataFrame is then saved to the corresponding CSV file using the `to\_csv` function.

***group\_emails\_by\_folder (self)***

The `group\_emails\_by\_folder` function is responsible for grouping the emails by folder. It returns a dictionary where each folder is associated with a list of emails.

Inside the function, the `emails` attribute of the `EmailManager` object is iterated to extract each email's details. For each email, the folder information is used to determine the corresponding folder in the dictionary. If the folder already exists in the dictionary, the email details are appended to the list of emails for that folder. If the folder doesn't exist, a new key is created in the dictionary, and the email details are stored as a list associated with that key.

This function is used internally to organize the emails before saving them to CSV files, ensuring that emails with the same folder are grouped together.

By utilizing these functions, the `EmailManager` class provides a convenient way to add and save emails, group them by folder, and store them in separate CSV files for later analysis and management.

**Usage Example**

The usage example demonstrates how to utilize the `EmailManager` class to collect and organize email details from the user.

1. The user is prompted to enter the necessary details of the emails, including the folder name (inbox, spam, important, social, or starred).

2. Within each folder, the user can enter the email ID, subject, date, and time. The program allows the user to add multiple emails to a single folder.

3. After entering the emails for a particular folder, the program asks if the user wants to enter another folder. If the user chooses to do so, they can repeat the process of adding emails to the new folder.

4. Once the user finishes entering all the emails, the program saves the emails to separate CSV files based on their folders. It also merges the CSV files into a single DataFrame, which can be accessed as `merged\_df`.

5. If the merging and saving process is successful, the program displays the first few rows of the merged DataFrame.

**Conclusion**

The Email Slicer Python Project provides a convenient solution for managing and organizing emails by categorizing them into different folders. The project allows users to enter email details and sort them based on specific criteria. The `EmailManager` class handles email operations, including adding emails, saving them to CSV files, and merging the files into a single DataFrame. By utilizing this project, users can efficiently manage their emails and extract valuable insights from the merged data for further analysis.