Name: Yifei Yu
Date: May 19th 2020
Topic: Documentation for outbound email automation programme

AutoEmail: Customised Routine Email Sender

Table of Contents

EXECUTIVE SUMMARY

Auto Email is a small command-line application that sends customised emails to a list of recipients at a tap of one's keyboard. The application is particularly useful for individuals who regularly email a large pool of recipients with customised content for each subgroups.

This application has a few advantages and disadvantages compared with email service providers' standard graphic interface. The main advantage is a smaller margin of error in delivered content due to centralisation of email content production and delivery processes. The application also significantly reduces overhead of time spent in clicking and scrolling in the conventional graphical interface. Most importantly, the application enables users not to be distracted by inbound emails while their objective is only to send emails.

The main drawback is an initial learning curve including database management and basic Python programming skill. For example, the list of recipients as well as their personal information is stored in a MySQL database, the most popular relational database format. Knowledge of querying the database is essential in operating Auto Email.

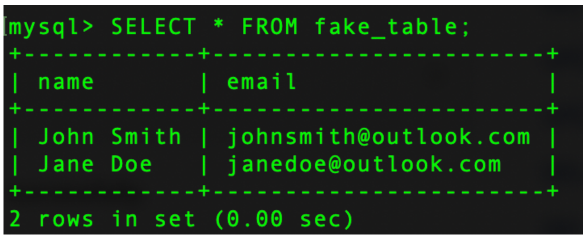
The following sections elaborate more in details. Section I describes a typical use case of Auto Email and serves as an user manual. Section II explains the programme structure and basic logic behind its construction. Section III cites areas of improvement to guide the programme's future development.

I. TYPIRCAL USE CASE

AutoEmail accepts recipient information, message content and mailbox account authentication information as inputs before delivering the emails and displaying delivery results. Users enter these inputs in three different locations including a *MySQL* database, a text file and the command line.

Building a *MySQL* database requires users to complete three steps including installation, database creation and data entries. Installation of *MySQL*[1] itself on *Linux* or *MacOS* operating systems can be done through a software package management system called *Homebrew*[2]. Both *Homebrew* and *MySQL* can be installed through typing commands in a shell prompt. In the case of a *MacOS* system, after typing "terminal" in Mac's Spotlight Search, users can type in and execute commands as instructed by the websites provided in this page's footnotes to install *MySQL*.

The next step of entering email recipients' information in *MySQL* is to create a database and complete queries for each individual recipient. A database is created by entering commands into an *MySQL* interface. The website called *w3schools.com* provides excellent instructions on how to create a database, create a table, insert a row and update a row. The goal is to create a table that includes all necessary information about email recipients. Below is an example of such table after it is created in *MySQL* and filled with values.



```
mysql> SELECT * FROM fake_table;
+------------+----------------------+
| name       | email                |
+------------+----------------------+
| John Smith | johnsmith@outlook.com |
| Jane Doe   | janedoe@outlook.com   |
+------------+----------------------+
2 rows in set (0.00 sec)
```

Fig.1: Example of a table within a MySQL database

The two columns "name" and "email" are essential for *AutoEmail* because an email has to be addressed to someone and to be sent somewhere. After entering information about email

[1] Link to MySQL's installation instructions through Homebrew:
https://medium.com/@youngstone89/how-to-install-mysql-on-mac-using-homebrew-e6f977b535ed
[2] Link to Homebrew's installation instructions:
https://brew.sh/

recipients, we move on to provide other input values for *AutoEmail*, to execute the programme and to review the results.

AutoEmail as a Python package[3] requires a few inputs as arguments. Below is a table including every input required to initiate the package.

database	name of the MySQL database containing recipient information
query	a SQL query statement retrieving the selected recipients' information
database_password	password to access the MySQL database
subject	subject line of the email to every recipient
body	body paragraphs of the email to every recipient
to_self	True/False binary to determine whether the email is a test email
self_query	a SQL query statement retrieving information of a test email recipient (i.e. the user's own mailbox)
own_email	email address of the user
own_email_password	password to access the user's own email address
name_col	name of the field that contains recipient names in the MySQL database
email_col	name of the filed that contains recipient email addresses in the MySQL database

Table 1: List of arguments of the object class "AutoEmail"

A Python package is typically imported and executed within a Python script. Below is a demonstration of a Python script that utilises *AutoEmail* to send a test email to the user him/herself.

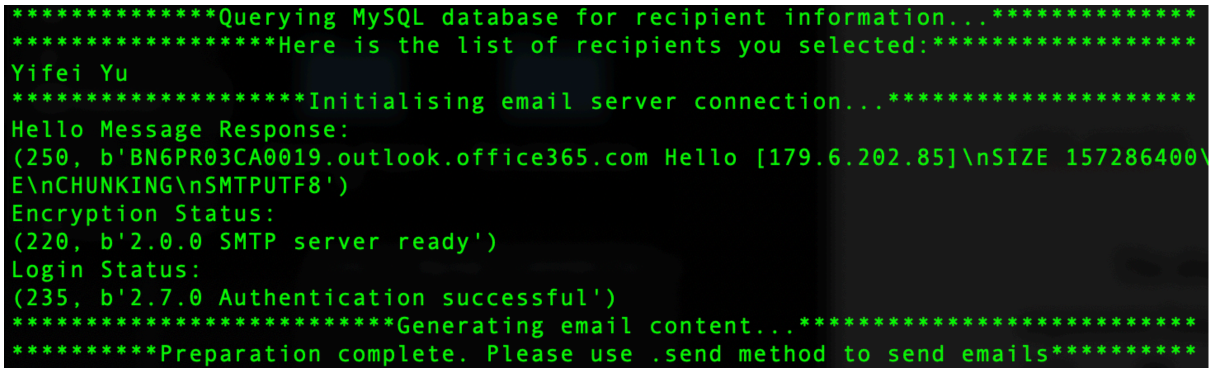
```
import AutoEmail

obj = AutoEmail(database = 'fake_db',
                 query = 'SELECT name, email FROM fake_table',
                 database_password = '123456',
                 subject = 'This is a test email',
                 body = 'This is the body of a test email',
                 to_self = True,
                 self_query = 'SELECT name, email FROM fake_table WHERE
                 name = \'John Smith\'',
                 own_email = 'johnsmith@outlook.edu',
                 own_email_password = '654321',
                 name_col = 'name',
                 email_col = 'email')
```

[3] Link to AutoEmail's Pypi project homepage:
https://pypi.org/project/AutoEmail/

```
obj.send()
```

Code Chunk 1: Instructions to initiate the main object class

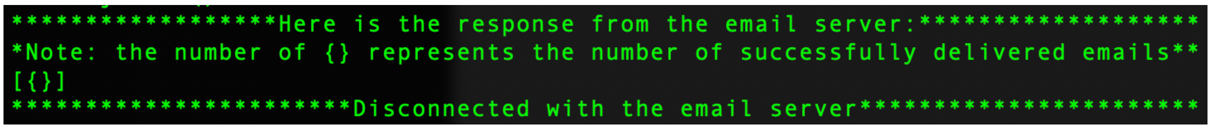
The above script initiates *AutoEmail*'s object class by supplying it with required arguments listed in *Table 1* and executes *AutoEmail*'s built-in method *.send()* to complete the delivery. The object initiation returns information about connection status as well as recipient names to allow users a final check before sending out the emails. Below is an example of *AutoEmail's* response if user authentication has been successful.



```
**************Querying MySQL database for recipient information...**************
******************Here is the list of recipients you selected:******************
Yifei Yu
********************Initialising email server connection...********************
Hello Message Response:
(250, b'BN6PR03CA0019.outlook.office365.com Hello [179.6.202.85]\nSIZE 157286400\
E\nCHUNKING\nSMTPUTF8')
Encryption Status:
(220, b'2.0.0 SMTP server ready')
Login Status:
(235, b'2.7.0 Authentication successful')
**************************Generating email content...**************************
**********Preparation complete. Please use .send method to send emails**********
```

Fig. 2: Response from "AutoEmail" indicating a successful authentication

The method *.send()* also returns a response indicating whether emails have been successfully delivered. Below is an example of such responses from the *.send()* method.



```
******************Here is the response from the email server:*******************
*Note: the number of {} represents the number of successfully delivered emails**
[{}]
***********************Disconnected with the email server***********************
```

Fig. 3: Response from ".send()" method indicating one successfully sent email

In the above response, the number of *{}* within *[]* indicates the number of successfully sent emails. In this example, we only sent the email to one recipient. Hence the number of *{}* is exactly one. If the programme fails to deliver an email, an error message will be included in *{}* as opposed to blanks.

After demonstrating a typical use case for the programme, the next section discusses briefly about the exact structure of the programme. Although the following technical breakdown is not essential to operate the programme, the details inform the application's future development as well as provide references for Section III when we explore areas of potential improvement for the application.

II. PROGRAMME STRUCTURE

The programme's structure consists of three parts which are database querying, content generation and delivery. All parts are enclosed in one single Python script by different functions within the same class object *AutoEmail*. Details of these individual components are described below.

Database querying allows the programme to fetch a user-selected list of recipient information from the *MySQL* database that contains every recipient's information. Mostly importantly, two columns including names and email addresses are required to be selected by the user to successfully execute the next two parts of *AutoEmail.* The first three arguments of the object class *AutoEmail*, **database**, **query** and **database_password** are inputs that enable the programme to complete the database query. While the name of the database and its password are straight-forward, query is more complicated especially for users who are unfamiliar with relationship database management system. While users are advised to reference the numerous *MySQL* instructions online, a typical query that satisfies *AutoEmail*'s requirement can be *SELECT name, email FROM fake_table WHERE name = 'John Smith'*.

Content generation constructs a subject line and a body for the email to each recipient and couples the content with the respective recipient's email address. *AutoEmail* achieves this feat by creating three Python dictionary objects, all of whose keys are names of the recipients. Python dictionary objects are associative arrays that create key-value pairs which are useful for us to establish relationship between recipients' names, email addresses and their individual email content. Within the source code, names of dictionaries storing information about email addresses, email subject lines and email bodies are *contact_list, subject_dict* and *content_dict*.

Delivery attempts to send packaged email content to each individual recipient and returns information about whether the attempt was successful or not. *AutoEmail* accomplishes this by first establishing a connection with the target email service provider server (i.e. Outlook server[4]) where all email exchanges take place. Then the programme unpackages data from the three previously constructed Python dictionaries to finally submit to the server. Error

[4] The exact name of this server is 'smtp-mail.outlook.com'

messages, if any, are displayed subsequently before the programme disconnects itself with the server.

III. LIMITATIONS AND POTENTIAL IMPROVEMENT

AutoEmail is an amateur project with the express goal of streamlining sending routine emails to a mostly constant list of recipients. Many limitations exist which either reduce the *AutoEmail* effectiveness by requiring users to have extensive programming knowledge or dampen user experiences through *AutoEmail*'s less intuitive interface. Paragraphs below describe and illustrate these limitations and discuss potential remedies that may be incorporated in *AutoEmail*'s future updates.

The first major limitation is *AutoEmail*'s inability to include text formatting, photo embedding and attachments in the current version. All body paragraphs at the moment can only be in the single spaced format with a mono font. Further development can create an additional interface in providing HTML (Hyper Text Markdown Language) as an input to AutoEmail's **body** argument to enable differently formatted text, embedded photos and attached files.

Another issue that hinders user experience is *AutoEmail*'s requirement on a well-defined *MySQL* database. Users are required to maintain the database which have to be completely error-free. Although the cost of time may be insignificant for applications in organisations of dozens of people, complexity quickly escalates as *AutoEmail* is applied in a larger scale. Future updates may provide additional support in drafting queries and validating information within the database to reduce the possibility of errors on the user's side.

The last area of improvement is an addition of inbound email management features. Current version of *AutoEmail* only partially automates the process of sending emails but filtering through received emails can also be an important feature. Further development of *AutoEmail* may include functions which automatically category inbound emails according to users' preference as prior inputs. Integration of inbound email management features has the possibility of turning *AutoEmail* into a one-stop shop for email management that no longer requires users to spend an excessive amount of time on clicking and scrolling.

IV. SUMMARY AND CONCLUSION

AutoEmail is a command-line application that automates the process of sending routine but slightly customised emails to a large group of recipients. Deployment of the application can be particularly profitable in small and medium enterprises that require a cheap yet effective solution to deliver information to its stakeholders.

AutoEmail functions by extracting information from a pre-defined database and delivering user-supplied text to recipients. Both authentication information and texts that users aim to send are entered into the programme as arguments for *AutoEmail*'s main class object.

The application's current state is best described as under development and has major drawbacks including database management overhead, support for photos and attachments, a user-friendly interface and user input validation protocol. Major development is required to upgrade the current application into an enterprise-class production-ready state.

Appendix

Python Script

```
'''
This package sends customised emails to selected recipients whose
information is stored in a local mysql database
'''

import numpy as np
import pandas as pd
import mysql.connector
import smtplib

class AutoEmail:
	def __init__(self, database, query, database_password, subject,
body, to_self, self_query, own_email, own_email_password, name_col =
'name', email_col = 'email'):
		self.own_email = own_email
		print('{0:*^80}'.format('Querying MySQL database for
recipient information...'))
		self.query_response, self.contact_list =
self.send_query(database, query, database_password, to_self,
self_query, name_col, email_col)
		print('{0:*^80}'.format('Initialising email server
connection...'))
		self.smtp_obj = self.initialisation(self.own_email,
own_email_password)
		print('{0:*^80}'.format('Generating email content...'))
		self.subject_dict, self.content_dict =
self.generate_content(self.contact_list, subject, body)
		print('{0:*^80}'.format('Preparation complete. Please
use .send method to send emails'))

	def send_query(self, database, query, password, to_self,
self_query, name_col, email_col):
		cnx = mysql.connector.connect(user = 'root', database =
database, password = password, auth_plugin = 'mysql_native_password')
		if to_self:
			query_response = pd.io.sql.read_sql(self_query, con =
cnx, index_col = None)
		else:
			query_response = pd.io.sql.read_sql(query, con = cnx,
index_col = None)
		cnx.close()
		contact_list = dict(zip(query_response.loc[:,
name_col].tolist(), query_response.loc[:, email_col].tolist()))
		print('{0:*^80}'.format('Here is the list of recipients you
selected:'))
		[print(full_name) for full_name in contact_list]
		return query_response, contact_list

	def initialisation(self, own_email, own_email_password):
		smtp_obj = smtplib.SMTP('smtp-mail.outlook.com', 587)
		print('Hello Message Response:\n' + str(smtp_obj.ehlo()))
		print('Encryption Status:\n' + str(smtp_obj.starttls()))
		print('Login Status:\n' + str(smtp_obj.login(own_email,
own_email_password)))
		return smtp_obj
```

```
      def generate_content(self, contact_list, subject, body):
            first_name_list, content_dict, subject_dict = [], {}, {}
            for full_name in contact_list:
                  first_name = full_name.split()[0]
                  content_dict[full_name] = 'Dear ' + first_name + ',\n'
+ body
                  subject_dict[full_name] = subject
            return subject_dict, content_dict

      def send(self):
            response = []
            for full_name in self.content_dict:
                  response.append(self.smtp_obj.sendmail(self.own_email,
self.contact_list[full_name], 'Subject: ' +
self.subject_dict[full_name] + '\n\n' + self.content_dict[full_name]))

            print('{0:*^80}'.format('Here is the response from the email
server:'))
            print('{0:*^80}'.format('Note: the number of {} represents
the number of successfully delivered emails'))
            print(response)
            self.smtp_obj.quit()
            print('{0:*^80}'.format('Disconnected with the email
server'))
```