Fumarco, Luca; Gaddis, S. Michael; Sarracino, Francesco; Snoddy, I.

Working Paper

Sendemails: An Automated Email Package with Multiple Applications

IZA Discussion Papers, No. 16163

Provided in Cooperation with:
IZA – Institute of Labor Economics

Suggested Citation: Fumarco, Luca; Gaddis, S. Michael; Sarracino, Francesco; Snoddy, I. (2023) : Sendemails: An Automated Email Package with Multiple Applications, IZA Discussion Papers, No. 16163, Institute of Labor Economics (IZA), Bonn

This Version is available at:
https://hdl.handle.net/10419/278861

I Z A Institute
of Labor Economics
Initiated by Deutsche Post Foundation

DISCUSSION PAPER SERIES

IZA DP No. 16163

Sendemails: An Automated Email Package with Multiple Applications

L. Fumarco
S. M. Gaddis
F. Sarracino
I. Snoddy

MAY 2023

I Z A Institute
of Labor Economics

Initiated by Deutsche Post Foundation

DISCUSSION PAPER SERIES

IZA DP No. 16163

Sendemails: An Automated Email Package with Multiple Applications

Luca Fumarco
Masaryk University, IZA and GLO

Francesco Sarracino
Research Division of STATEC and GLO

S. Michael Gaddis
University of California and Northwest Evaluation Association

I. Snoddy
Analysis Group Vancouver, British Columbia

MAY 2023

ISSN: 2365-9793

IZA – Institute of Labor Economics

Schaumburg-Lippe-Straße 5–9
53113 Bonn, Germany

Phone: +49-228-3894-0
Email: publications@iza.org

www.iza.org

ABSTRACT

Sendemails: An Automated Email Package with Multiple Applications*

Correspondence audits are a popular method to examine discrimination in a causal framework. However, they often require sending hundreds or thousands of emails to subjects. The sendemails package allows users to automatically send emails with Stata through PowerShell, which is open-source and cross-platform. Researchers can use this package to perform several email tasks, such as contacting students or colleagues with standardized messages. Additionally, researchers can perform more complex tasks that entail sending randomized messages with multiple attachments from multiple accounts, tasks that are often necessary to conduct correspondence audit tests. This paper introduces the command and illustrates multiple examples of its application.

JEL Classification: C8
Keywords: sendemails, PowerShell, email

Corresponding author:
Luca Fumarco
Masaryk University
Tvrdého 12
60200 Brno
Czech Republic
E-mail: luca.fumarco@econ.muni.cz

* This manuscript is currently under review at Stata Journal.

1 Introduction

This article introduces a new Stata package, **sendemails**, which allows instructors, researchers, or students to send emails through Stata, to the benefit of their pedagogical or scientific tasks.

Stata is a powerful software for data science that can be adapted to send messages. To present, only limited and sparsely documented resources have been provided for this purpose. Some contributed do-files allow users to send messages from specific email providers[1], but their customization can be complicated and time-consuming for people with limited programming skills. Some users have prepared dedicated Stata packages intending to simplify the process of sending emails; a good example is `psemail` by Xuan Zhang and colleagues (2013). However, these early packages have some limitations; for example, they either do not allow html input,[2] they do not allow sending multiple attachments or do not include standard email options, such as delivery notifications and priority.

sendemails overcomes these limitations. This is a simple general and flexible tool, which can be used by anyone. It allows several useful applications such as sending several emails to students or colleagues. For example, the user can send personalized emails to students with their exam grades, or to send a message to people in a mailing list. Our package can send one email at a time, within a short period, thus overcoming the limit on the maximum number of recipients per email usually set by the email provider. Outlook, for instance, imposes a limit of 500 recipients per message. With **sendemails** the only constraint is the maximum quantity of recipients allowed per day which, in the case of Outlook, is 5,000. Moreover, **sendemails**

[1] For example, there is a Stata script to send emails with an Outlook application through VBScript (http://www.wmatsuoka.com/stata/sending-an-email-with-stata-outlook-edition) or a script to send emails with an Outlook application invoked through Stata's shell (https://www.stata.com/statalist/archive/2013-01/msg01201.html).
[2] This limitation implies that is not possible to use italic or bold characters, for example.

allows sending emails with attachments (e.g., tables and figures) after the completion of tasks that can take several days to be carried out. In this regard, **sendemails** offers features similar to other Stata packages, such as `sendtoslack` (which sends messages through Slack and is available here: http://fmwww.bc.edu/repec/bocode/s/sendtoslack.ado) and `statapush` (which sends messages through Pushbullet, Pushover, or IFTT, and is available here: https://github.com/wschpero/statapush).[3]

sendemails allows users to send messages from different addresses, which might be useful for a variety of reasons. For example, for different projects, the user could correspond with different emails (e.g., a personal email address, the former affiliation address, and the current affiliation address). For this, **sendemails** can be very useful to conduct online experiments, such as correspondence audit tests. For more details and an example of the specific application of **sendemails** to correspondence audit tests, please refer to Appendix 2.

sendemails requires **tknz** to be installed. This package tokenizes strings into tokens and stores the results in the local macros.

In the following sections, we discuss the details of this command and provide examples to help the reader better understand the proper usage and full potential of **sendemails**.

2 A Command for Standardizing and Automatically Sending Email: sendemails

2.1 Syntax

sendemails uses Windows PowerShell to send emails.

This is the syntax of **sendemails**:

[3] Readers may also be interested in packages that enable Stata to reproduce an audio file through Windows Media Player when a task is concluded, such as milito22 (https://bit.ly/3xwU47v).

```
sendemails recipient@email.com [if] [, subject(string)
body(string) attachment(string) folder(string) html(string)
paragraphs(string) htmltxt(string) psloc(string)
mailps(string) smtpport(string) smtpsserver(string)
from(string) sleep(string) ufile(string) pfile(string)
cc(string) bcc(string) notification(string) priority(string)
encode(string) pstimeout(string) nossl report
erroroption(string) errormessage(string)]
```

The argument of the command **`sendemails`** is:

`recipient@email.com`.

The argument specifies the recipient of the email, and it cannot be omitted. When multiple recipients are included, they have to be separated by a semicolon, without any blank space (e.g., “luca@myemail.it; stefano@hisemail.co.uk" is not allowed by the underlying PowerShell `cdmlet`, more details are provided below).

2.2 Options

Options are organized into five subsections.

Main options:

`subject(string)` declares the email subject. It must be specified.

`body(string)` specifies the body of the email to be sent. The body should be input as a single string with blocks of text being parsed by “`|`”. Each substring separated by “`|`” will be numbered and modified using html wrappers if specified. Text inputted as "`line1 |`

`line2 | line3"` will be treated as 3 separate blocks of text and numbered 1, 2, 3. It cannot be invoked when `htmltxt(string)` is specified.

`from(string)` is the name of the sender. The default parameter value is the email username.

`ufile(string)` gives the location and filename of the user's email address saved as plain text in a `.txt` file. The default location is the working directory. The file extension is optional. The file name must be specified.

`pfile(string)` gives the location and filename of the user's password saved as plain text in a `.txt` file; the default location is the working directory. The file extension is optional. The file name must be specified. It should be noted that the user does not have to enter the actual password and username inside `ufile(string)` and `pfile(string)`. The user should rather write locations and filenames of the two txt-files from which **`sendemails`** will extract username and password.

`mailps(string)` gives the name of the `.ps1` file, default is "`mailps.ps1`". `.ps1` is the script that contains PowerShell commands, and it can be opened with any `.txt` file editor. This file can be used to double-check the details of the message being sent (e.g., subject, body and line breaks, recipient).

`psloc(string)` gives the folder location where the `.ps1` file will be saved, the default location is the working directory.

Attachment options:

`attachment(string)` gives the location and name of the files to be included as attachments to the email. File extension must be included; in alternative, if there is no file with the same name, the user can type "`.*`" in place of the extension. If the location is not

included, Stata looks for the attachment in the working directory. When multiple files are included, they have to be separated by a semicolon, without any blank space (e.g., "lucapaper.pdf; Stefano.*" uses the semicolon, but includes a blank space so it is not allowed).

folder(string) gives the folder location of all the attachments the user wants to send at once. If the user wants all of the files in the folder, the user must type "*.*" following the folder name; assuming the folder is called "robe" then string = "working_directory_path\robe*.*". If the user wants only those files with a particular extension, the second * must be substituted by the extension. For example, if the user wants to attach all the pdf's from a folder containing also .dta's, .doc's, etc…, the user should type "*.pdf"; assuming the folder is called "robe" then string = "working_directory_path\robe*.pdf". The user should note that if there are files with the same name, but different extensions in the same folder, and the user wants to attach them all, then string = "working_directory_path\robe\file_name.*" could be written. The user cannot specify attachment(string) when folder(string) is specified.

Formatting options:

html(string) provides the html to be included in the email and that modifies the appearance of the text. As each block of text parsed by "|" in the body is numbered, the html code should be provided as wrappers around these numbers. Users should input a string where html wrappers are placed around numbers, with each number representing a substring of the body. For example, "1 2 3 4" makes all text in substrings 1

and 2 italic and text in substring 2 bold. If html is specified, it must include a number for every substring. If there are 4 inputted substrings and html takes input "`<b> 1 </b> 2 3`" then the fourth string will be missing from the email. The user can create paragraphs using html or using par. This option cannot be specified when `body(string)` is empty.

`paragraph(string)` provides paragraph breaks between substrings parsed by "`|`" in body. The input to this option should take the format of a numbered list, for example, `par(1 3 5)` creates a new paragraph following substrings 1, 3, and 5. This option cannot be specified when `body(string)` is empty.

`htmltxt(string)` provides the location of the txt file with the html message to be included in the email and that modifies the appearance of the text. It cannot be specified when `body(string)` is specified.

`encode(string)` is the encoding option. PowerShell allows the following parameters: ASCII, BigEndianUnicode, BigEndianUTF32, OEM, Unicode (default), UTF7, UTF8, UTF8BOM, UTF8NoBOM, UTF32. Beginning with PowerShell 6.2, `encode()` also allows numeric IDs of registered code pages or string names of registered code pages. For more information, see the .NET documentation for Encoding. PowerShell allows abbreviated parameter names until the parameter is no longer unambiguous; cases do not matter.

Sending options:

`smtpport(string)` gives the smtp port number. The default parameter value is the Outlook port.

`smtpsserver(string)` gives the smtp server address. The default parameter value is the Outlook server.

`cc(string)` is the email address included as a cc to the email. When multiple cc's are included, they have to be separated by a semicolon, without any blank space (e.g., "luca@myemail.it; stefano@hisemail.co.uk" is not allowed).

`bcc(string)` is the email address included as a bcc to the email. When multiple bcc's are included, they must be separated by a semicolon, without any blank space (e.g., "luca@myemail.it; stefano@hisemail.co.uk" is not allowed).

`sleep(string)` is the time gap between multiple emails sent in sequence (e.g., through a loop). The default parameter value is 3.000 milliseconds (i.e., 3 seconds).

`notification(string)` is the notification option. It provides feedback to the author about delivery status. As allowed by PowerShell, it can be set to: `None` (default) , `Never`, `OnSuccess`, `OnFailure`, `Delay`, or any combination of the last three parameter values (e.g. `OnSuccess, Delay`). PowerShell allows abbreviated parameter values until the parameter is no longer unambiguous; cases do not matter.

`priority(string)` is the priority option. PowerShell allows the following options: `Normal` (default), `High`, or `Low`. PowerShell allows abbreviated parameter values until the parameter is no longer unambiguous; cases do not matter.

`nossl` specifies that the Secure Sockets Layer (SSL) to establish a connection is not used.

Reporting options:

`report` tells Stata to display a summary of email with sender and receiver(s), email options (i.e., priority, notification, encode), optional metadata (i.e., psloc, mailps, PowerShell

timeout, error option), time and date of the email. When email options and metadata are not set by the user, this report lists the default options.

`erroroption(string)` tells PowerShell what to do in case of an error. The following options are allowed: `Stop`, `Inquire`, or `Continue` (default). Differently from other PowerShell options, no abbreviations or different combinations of capital and lowercase letters are allowed to prevent mistakes; the only allowed exception is the usage of lowercase letters only. When `Continue` is chosen, if PowerShell detects an error, it does not stop and it does not display any message. When `Stop` is chosen, if PowerShell detects an error, it stops and displays an error until the user manually closes the PowerShell window. When `Inquire` is chosen, if PowerShell detects an error, it stops and inquires on what to do, that is, either `Stop` or `Continue`—additional parameters being provided are useless in this context, but are provided by default by PowerShell and cannot be omitted by us; after the user responds the inquiry, PowerShell automatically closes its window.

`errormessage(string)` is the name of the txt file where the error message should be written by PowerShell. The error message transcribed in that file is the same as the one that would be displayed in the PowerShell window. The default name of the txt is 'errormessage'. While PowerShell window does not display the error message when `erroroption()` is `Continue`, the error message will be visible in this txt. The location of this error message is psloc. `pstimeout(string)` is the timeout option that tells how long PowerShell should wait before closing its window after the **`sendemail`**s command has run. If an error occurred and `erroroption` is `Stop`, the window will close after 3' as by PS default or after the user's closes it—the timeout

option is irrelevant in this case. If an error occurred and `erroroption` is `Continue`, the window will close after the default five seconds or after the number of seconds selected by the user. If an error occurred and `erroroption` is `Inquire`, the window will close following the user's choice after the default number of seconds or after the number of seconds selected by the user.

3 Technical Details

PowerShell was originally a Windows component exclusively; however, on August 2016 it was made open-source and cross-platform. If you do not have already PowerShell (e.g., if you have an Apple computer), you can download it here: https://github.com/PowerShell/PowerShell.

Emails are sent using the `Send-MailMessage cmdlet` (information here: https://msdn.microsoft.com/en-us/powershell/reference/5.1/microsoft.powershell.utility/send-mailmessage).

`sendemails` sends emails using SMTP (Simple Mail Transfer Protocol). Users' credentials, the recipient's email address, and features to be included in the email are captured by the program. A `.ps1` script is created and runs using PowerShell. This file is stored on the user's system in a specified location along with the user's credentials.

`sendemails` works only with email providers that allow access from external apps. By default, the smtp settings are set for the use of Outlook accounts. These settings can be modified to work for other email providers. For instance, to send an email using zoho.com the `smtpserver` should be set to "`smtp.zoho.com`" and the `smtpport` to "`587.`" Different email accounts may require users to modify settings before the use of smtp. For instance, Gmail requires that users allow access to less secure apps before an email can be sent using smtp via

PowerShell (details on this are in Appendix 1). We are aware of only one email provider that does not allow smtp protocol and thus **sendemails**; this is Office365, since December 31, 2022. The speed of email delivery may also differ across services. Appendix 1 reports smtp servers and ports for some of the most popular email providers, in Table A.1.

Users who have not used PowerShell before should note that by default they may not have sufficient administrative privileges to perform the operations in **sendemails**. First-time users should open PowerShell as administrators and type "`Set-ExecutionPolicy RemoteSigned.`" This command must be given only the first time that **sendemails** is run on the machine; it specifies that scripts created on the current system and files with a digital signature may be executed.

There are two alternative short routes for this execution policy change. The choice between which route to follow depends on the user's needs and administrator rights. The first route is the fastest one. The user can open PowerShell from within Stata by typing: `!powershell -noexit -command "Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser".` This policy is not changed as an administrator; thus, it will be valid only for the current user. Other users of the same machine should change their execution policy too. Notice that some institutes or companies may not grant local administrator rights; in this case, it is necessary to request access as an administrator to the local IT staff. The second alternative route requires a minimum direct interaction with PowerShell (i.e., from outside of Stata), but it changes the execution policy for all users of the machine. The user can open PowerShell as an administrator from within Stata by typing: `!powershell -noexit -command "start-process powershell -verb runas"` which opens a window called "User Account Control" asking to allow

PowerShell to make changes to the device. The user should select "yes." Afterwards, the command window named "Administrator: PowerShell" opens. At the end of the first line—where the last character is '>'—the following command should be typed: `Set-ExecutionPolicy RemoteSigned -Force` and then the Enter key should be pressed. The option `-Force` tells PowerShell to skip the confirmation prompt, that is, it does not ask the user to confirm the change to the execution policy.

To check and reset the execution policy from within Stata, two commands are available. First, `!powershell -noexit -command "Get-ExecutionPolicy"` prompts PowerShell to display the current policy. Second, `!powershell -noexit -command "Set-ExecutionPolicy -ExecutionPolicy Default -Scope CurrentUser"` sets the policy back into being `Restricted`.

Owed to the configuration of `the Send-MailMessage cmdlet`, the way locations and file names can be written follows some rules. There cannot be blank spaces in: (i) `attachment(string)`, (ii) in `folder(string)`, (iii) between multiple main recipients, (iv) between additional recipients in both `cc(string)` and `bcc(string)`.

In any option that may include a location, this location cannot include blank spaces, even between quotes in Stata (i.e., `psloc("C:\Users\Documents\Stata Jour Paper")` is not allowed). Some options are mandatory, namely `ufile(string)`, `pfile(string)`, and `subject(string)`.

Multiple main recipients as well as additional recipients in both `cc(string)` and `bcc(string)` must be separated by a semicolon without blank spaces.

Parameter values in `notification(string)` must be separated by a comma and may include blank spaces.

Parameter values of `notification(string)`, `encode(string)`, and `priority(string)` accept parameters abbreviations as well until the parameter is no longer unambiguous; moreover, cases do not matter.

4 Setting up the material for the examples

Before starting the series of examples, the user should create two one-word txt-files: one file includes the actual password and we call it "`password`", while the other file includes the username and we call it "`username`"—which usually equals the email address itself, referring to the email address used for sending messages. We recommend the user writes the txt-files from outside Stata to protect the user's personal information in case the do-file with the **`sendemails`** command had to be shared with someone else. The txt-files can be produced, for example, directly on Notepad or Notepad++ and stored in the working directory. These two txt-files are used in Example 5.1. In alternative, the user could produce password and username files from within Stata. This is a minimal working example:

```
     cd "working_directory_path\"
     file open myfile using "password.txt", write replace
     file write myfile "mypassword" //where mypassword is the
user's actual password
     file close myfile
     file open myfile using "username.txt", write replace
     file write myfile "myemail" //where myemail is the user's
actual email address
     file close myfile
```

Note that the same process of storing email address details somewhere in the machine is used in `psemail;` instead of having password and username in two separate txt-files, they are in macro globals in the `profile.do`. An example of how users can take advantage of macro globals stored in the `profile.do` when using `sendemails` is presented in Section 6. Note

that, because of `psemail` nature, the user can store and use only one email address, while `sendemails` allows details on several senders' email addresses to be stored and used, depending on the needs.

We recommend the user to load the example dataset, which is (partially) used from Example 5.2 onward. The dataset is called "`excelstatajexamples.xls`" and it contains four lines, from different senders to different receivers, with several personalized components of the message. The user should personalize the dataset so that the string variable called "email" contains those emails used in the examples in this and the following section. The string variable called "server" and the numeric variable called "port" should include the parameters that correspond to "email." The string variable called "receiver" should contain the recipient address. Finally, the variables "password" and "username" are already populated with strings corresponding to names of the txt-files that contain the actual password and the email username for the sending email addresses used in Section 6. To replicate the examples in Section 6, the user needs to create four additional txt-files.

The user should personalize similarly the anonymized do file "`examplesendemails–anonymized.do.`"

Once the txt-files are ready, the excel and the do-files are personalized, this do-file can run all the examples from Example 5.1, including the one in Appendix 2. Remember also to set up administrative privileges, as discussed in Section 3.

5 Examples

5.1 Example 1: Sending one message

Let us send a one-liner email using the email provider Outlook. Let us give Stata the command:

```
sendemails try.send.emails2@gmail.com , ///
body(ciao amico) sub(hi)  ///
```

```
ufile(USERNAME2) ///
pfile(PASSWORD2-APPPWD) ///
smtpport(587) smtps(smtp.gmail.com) ///
psloc(.\)
```

This command will create an email that looks like this:

```
ciao amico
```

The email subject is "hi".

The user with an Outlook account can try this same example while omitting `smtpport` and `smptserver` options; the email will be sent anyway, because of the default settings.

Stata should be open whenever an email is to be sent.

5.2 Example 2: Sending one message to more than one email address

This example sends one email message as in Example 1; however, the body is personalized and the email is sent to several receivers from several sending addresses. We additionally set the `.ps1` location in a different folder, the sleep time, and the name of the `.ps1` files. Let us give Stata the command:

```
import excel .\excelstatajexamples.xls, firstrow clear

local N=_N
forval i =1(1) `N' {

local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

sendemails `receiver', ///
body(ciao `nome') sub(hi)  ///
smtpport(`port') ///
smtpserver(`server') ///
ufile(`username') ///
pfile(`password') ///
psloc(.\) sleep(3000) ///
```

```
mailps("`receiver'`nome'B")
   }
```

This will create an email that looks like this:

```
ciao FRIEND'S NAME
```

Note that `mailps` sets a name for the `.ps1` file; this name is composed of the receiver's email and name, and a letter. This file, as well as those from the next examples, are all included in the working directory.

5.3 Example 3: Sending one message to more than one email address, with same cc

This example is the same as Example 2, except that one email in cc is included as well. Let us give Stata the command:

```
local N=_N
forval i =1(1) `N' {

local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

sendemails `receiver', ///
body(ciao `nome') sub(hi)  ///
smtpport(`port') ///
smtpserver(`server') ///
ufile(`username') ///
pfile(`password') ///
cc(CCEMAIL@provider.com) ///
psloc(.\) sleep(3000) ///
mailps("`receiver'`nome'C")
```

5.4 Example 4: Sending one message to more than one email address, with same cc and same bcc

This example is the same as Example 3, except that we add an email address in bcc. Let us give Stata the command:

```
local N=_N
forval i =1(1) `N' {

local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

sendemails `receiver', ///
body(ciao `nome') sub(hi)  ///
smtpport(`port') ///
smtpserver(`server') ///
ufile(`username') ///
pfile(`password') ///
cc(CCEMAIL@provider.com) ///
bcc(BCCEMAIL@provider.com) ///
psloc(.\) sleep(3000) ///
mailps("`receiver'`nome'D")

}
```

5.5 Example 5: Sending one message to more than one email address, with same cc and same bcc, and one attachment

This example is the same as Example 4, except that one file—the extension is irrelevant—is attached to the email. Let us give Stata the command:

```
local N=_N
forval i =1(1) `N' {

local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

sendemails `receiver', ///
body(ciao `nome') sub(hi)  ///
smtpport(`port') ///
smtpserver(`server') ///
ufile(`username') ///
pfile(`password') ///
```

```
cc(CCEMAIL@provider.com) ///
bcc(BCCEMAIL@provider.com) ///
psloc(.\)  ///
attachment(ecselfile2.*) ///
sleep(3000) ///
mailps("`receiver'`nome'E")
}
```

Note that there is only one file called `ecselfile2`, so the extension is not needed.

5.6 Example 6: Sending one message to more than one email address, with same cc and same bcc, and all the files from a folder

This example is the same as Example 4, except that all the files from a folder are attached to the email—we call the folder `ATTFOLDER`, it includes five files with different names and extensions. Let us give Stata the command:

```
local N=_N
forval i =1(1) `N' {
local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

sendemails `receiver',   ///
body(ciao `nome') sub(hi)     ///
smtpport(`port')         ///
smtpserver(`server')     ///
ufile(`username') ///
pfile(`password') ///
cc(CCEMAIL@provider.com) ///
bcc(BCCEMAIL@provider.com) ///
psloc(.\) ///
folder(.\ATTFOLDER\*.*)  ///
sleep(3000)              ///
mailps("`receiver'`nome'F")
}
```

What follows is a slightly modified version of Example 6. The difference is that now all the files with the same extension are attached from the folder `ATTFOLDER`. Let us give Stata the command:

```
local N=_N
forval i =1(1) `N' {
local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

sendemails `receiver',   ///
body(ciao `nome') sub(hi)     ///
smtpport(`port')         ///
smtpserver(`server')     ///
ufile(`username') ///
pfile(`password') ///
cc(CCEMAIL@provider.com) ///
bcc(BCCEMAIL@provider.com) ///
psloc(.\) /// what follows is the option to attach all the
files from a folder
folder(.\ATTFOLDER\*.xls)     ///
sleep(3000)              ///
mailps("`receiver'`nome'F")
}
```

5.7 Example 7: Sending one message, in three paragraphs, to more than one email address, with same cc and same bcc, and one attachment.

This example is the same as Example 5, except that one long email body is used, and is divided into three paragraphs using "|" as a delimiter. Let us give Stata the command:

```
local N=_N
forval i =1(1) `N' {

local nome = nome[`i']
local username = username[`i']
local password = password[`i']
local port = port[`i']
local server = server[`i']
local receiver = receiver[`i']

local body "In this study, I show that with the appropriate
experimental strategy, a correspondence test can be adapted to
investigate disability discrimination in the rental housing
market. | I focus on discrimination against blind tenants
assisted by guide dogs in Italy and obtain very robust results.
```

The utilization of three fictitious household tenants (that is, a married couple, a married couple with a blind wife who owns a guide dog, and a married couple where the wife is normal sighted and owns a pet dog) allows me to investigate whether discrimination is due to the blindness or to the guide dog. I find that apartment owners discriminate blind tenants because of the presence of the guide dog alone. | According to the Italian law, this is indirect discrimination, which in the US corresponds to the refusal to provide reasonable accommodation."

```
sendemails `receiver', ///
body(`body') sub(hi)  para(1 2) ///
smtpport(`port') ///
smtpserver(`server') ///
ufile(`username') ///
pfile(`password') ///
cc(CCEMAIL@provider.com) ///
bcc(BCCEMAIL@provider.com) ///
attachment("ecselfile2.*") ///
psloc(.\) sleep(3000) ///
mailps("`receiver'`nome'G")
}
```

This example produces an email text that looks as follows:

"In this study, I show that with the appropriate experimental strategy, a correspondence test can be adapted to investigate disability discrimination in the rental housing market.

I focus on discrimination against blind tenants assisted by guide dogs in Italy and obtain very robust results. The utilization of three fictitious household tenants (that is, a married couple, a married couple with a blind wife who owns a guide dog, and a married couple where the wife is normal sighted and owns a pet dog) allows me to investigate whether discrimination is due to the blindness or to the guide dog. I

find that apartment owners discriminate blind tenants because of the presence of the guide dog alone.

According to the Italian law, this is indirect discrimination, which in the US corresponds to the refusal to provide reasonable accommodation."

This is the abstract from Fumarco (2017), with line breaks.

5.8 Example 8: Sending an email with html bold words

This example is similar to Example 1, but it is sent from only one email address, the message is shorter, and some words are in bold. Let us give Stata the command:

```
     local body "hello friend, | How are you doing? | Best,
Friend"
     sendemails try.send.emails2@gmail.com, body(`body') sub(hi)
html(1 <b>2</b> 3)  ///
     smtpport(587) smtpserver(smtp.gmail.com) ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
     psloc(.\)
```

This will create an email that looks like this:

hello friend, **How are you doing?** Best, Friend

Note that if we erroneously specified *html(1 2)* the output would be:

hello friend, **How are you doing?**

5.9 Example 9: Sending an email with html paragraphs

This example follows Example 8, but the paragraphs are generated with two alternative methods: first, using html commands, and second, using the par(paragraphs) option.

The following uses of the **sendemails** command are equivalent:

```
     local body "hello friend, | How are you doing? | Best,
Friend"
     sendemails  try.send.emails2@gmail.com, body(`body') sub(hi)
html(1 2<br><br> 3)  ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
     smtpport(587) smtpserver(smtp.gmail.com) ///
     psloc(.\)

     local body "hello friend, | How are you doing? | Best,
Friend"
     sendemails  try.send.emails2@gmail.com, body(`body') sub(hi)
para(2)  ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
     smtpport(587) smtpserver(smtp.gmail.com) ///
     psloc(.\)
```

Both commands give the same result:

```
    hello friend, How are you doing?
    Best, Friend
```

5.10 Example 10: Sending an email using the html content included in an external document named textmessage.txt

This example shows how to send an email with html formatting. The difference compared to Examples 8 and 9 is that now the content and html formatting are included in an external document called "textmessage.txt" that is invoked using the `htmltxt()` option.

```
     sendemails  try.send.emails2@gmail.com, sub(hi) /// what
follows is the option to include an external txt file with the
content of the message and html formatting
     htmltxt(.\textmessage.txt) ///
     ufile(USERNAME2) pfile(PASSWORD2-APPPWD) ///
     smtpport(587) ///
     smtpserver(smtp.gmail.com)
```

Notice that `htmltxt(string)` cannot be specified together with the option `body(string)`.

5.11 Example 11: Sending an email with the results of a Stata program

This example shows how to send via email the output of a Stata command at the end of a routine.

```
     sysuse auto.dta, clear
     forvalues i = 1/5{
     regress price mpg if rep78 == `i'
     parmest, label list(parm label estimate) saving(./est`i'.dta,
replace)
     }
     use ./est5.dta, clear
     append using est1.dta  est2.dta  est3.dta  est4.dta
     save ./results.dta, replace

     sendemails try.send.emails2@gmail.com,  ///
     body(Please, find attached the results of your regressions)
sub(Regression results)       ///
     smtpport(587)       ///
     smtpserver(smtp.gmail.com)    ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
     psloc(.\) /// what follows attaches a list of files with the
results.
     attachment(results.dta)  ///
     sleep(3000)              ///
     mailps(try.send.emails2-Results)
```

It is straightforward to include **`sendemails`** in the loop and let it send the result of each estimate separately as the loop progresses. An example follows.

```
     sysuse auto.dta, clear
     forvalues i = 1/5{
     regress price mpg if rep78 == `i'
     parmest, label list(parm label estimate)
saving(./est`i'.dta, replace)

     sendemails try.send.emails2@gmail.com,  ///
     body(Please, find attached the results of your regressions)
sub(Regression results "`i'")      ///
     smtpport(587)       ///
     smtpserver(smtp.gmail.com)    ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
```

```
     psloc(.\) /// what follows attaches a list of files with
the results.
     attachment(./est`i'.dta)      ///
     sleep(3000)              ///
     mailps(try.send.emails2-est`i')
     }
```

5.12 Example 12: Inspecting the output in case of an error

Let us consider the case when the user makes a mistake, and—hypothetically—misspells the receiver's email address. The option `report` prompts Stata to provide a report on the details of the last sent email. Inspecting the content of the report reveals that the receiver's email is incomplete. An example follows.

```
     sendemails try.send.emails2@gmail. , /// input the email of
the receiver with an error
     body(ciao amico) sub(hi)  ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
     smtpport(587) smtps(smtp.gmail.com) ///
     psloc(.\) /// the following option prompts Stata to report
the details of the sent email
     report
```

This is the output of `report`.

```
-------------------------------------------------(Email details
- Start)
Email sender and receiver(s)
From:                     try.send.emails1@gmail.com
To:                       try.send.emails2@gmail.
Cc:
Bcc:

Email options
Priority:                 Normal
Notification:             None
Encode:                   Unicode

Email optional metadata
Ps location:              .\
Mailps file:              mailps.ps1
Powershell waiting time:  1 seconds
```

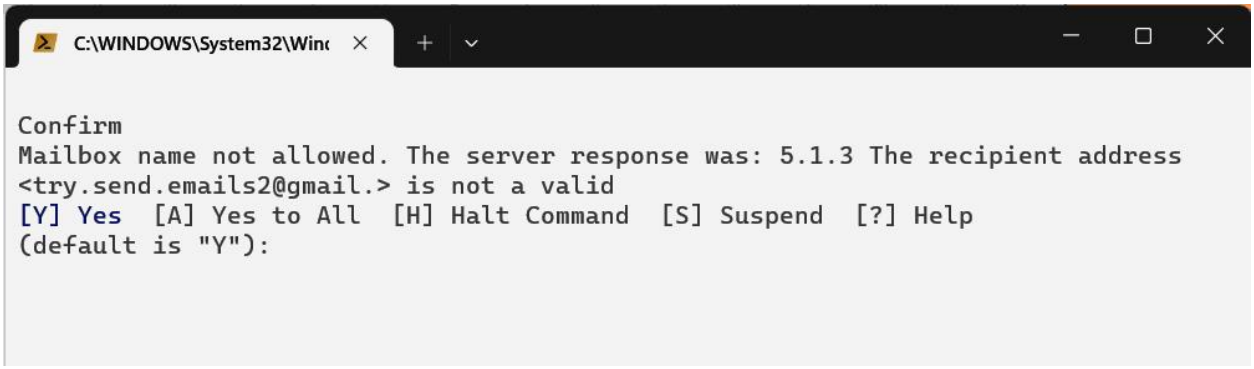
```
Powershell error option:  Continue

Time and date
Email sent at 17:37:16 on 17 Apr 2023
------------------------------------------------(Email details
- End)
```

Besides asking for a report, the user can prompt PowerShell to ask the user what to do in case of an error. An example follows.

```
     sendemails try.send.emails2@gmail. , /// input the email of
the receiver with an error
     body(ciao amico) sub(hi)  ///
     ufile(USERNAME2) ///
     pfile(PASSWORD2-APPPWD) ///
     smtpport(587) smtps(smtp.gmail.com) ///
     psloc(.\) /// the following option prompts PowerShell to ask
the user what to do in case of an error
     erroroption(inquire)
```

In this case, Stata executes the command, invokes PowerShell which reports an error, and asks the user how to proceed, see Figure 1.

Figure 1. PowerShell encounters an error and inquires how to proceed.



The user can choose to continue and disregard the error or to stop or suspend the command. The last option, "`help,`" provides details about what each option entails, namely:

```
     Y - Continue with only the next step of the operation.
     A - Continue with all the steps of the operation.
```

```
H - Stop this command.
S - Pause the current pipeline and return to the command
prompt. Type "exit" to resume the pipeline.
```

6 Automatize own details loading

If the user is always using the same email address(es) to send messages, we suggest writing a global macro in `profile.do`, which will be automatically run in the background by Stata whenever the software is opened. Let us write the macro and pretend it is saved in the profile.do:

```
//type the following macro in profile.do
global EmailDetails `"ufile(USERNAME2) pfile(PASSWORD2-
APPPWD) smtpport(587) smtpserver(smtp.gmail.com)"'

//then sendemails could be written as follows
sendemails try.send.emails2@gmail.com, ///
body(ciao amico) sub(hi) $EmailDetails ///
psloc(.\)
```

Now, the command for Example 1 can be written as follows:

```
sendemails try.send.emails2@gmail.com, b(ciao amico) s(hi)
$EmailDetails psloc(.\)
```

It is worth noting two things. First, the way this command works allows the user to write in `profile.do` details of different sending emails. For example, the user could have three different email addresses for sending emails, in this case, `profile.do` would contain three global macros, such as `EmailDetails1, EmailDetails2, and EmailDetails3`. Second, the global macro(s) (or the `profile.do` file) can be enriched with additional, frequently used options to minimize the number of options invoked each time. In the following example, some common options about message priority and error notification are added:

```
global EmailDetails `"ufile(USERNAME2) pfile(PASSWORD2-
APPPWD) smtpport(587) smtpserver(smtp.gmail.com)  pstimeout(5)
priority(High) report notif(OnSuccess, Delay)"'
```

7 Verify possible errors and what emails have been sent

Many things could go wrong in the process of sending emails; we identified five types of errors:

(i) Errors detectable by the email provider. For example, the recipient address might contain a non-existent username (e.g., verdghdjsp@hotmail.it). Strictly speaking, this is not a mistake either in the underlying PowerShell script or in the Stata script, so it will not be detected by them. In this case, an automatic email with an error message would be sent to the delivering address by the email provider of the receiver. Another example of an "error" detectable by the email provider is that the receiver's mailbox is full or no longer active.

(ii) Port and server errors. There could be an error in how the port or the server is typed, this is not a mistake in either the underlying PowerShell script or the Stata script. This error cannot be detected by either PowerShell or Stata—to the best of our knowledge. It is not even an error detectable by the email provider, because the email provider cannot be contacted in the first place if port and/or server are wrong. In this case, the PowerShell operation cannot be concluded, so it reaches the default operation timeout of 3' (note that this parameter does not correspond to the `pstimeout()` option and cannot be easily changed by PowerShell users with a simple script). No error message is thus delivered by the email provider, by PowerShell, or by Stata.

(iii) Writing Stata command.

(iv) Errors in the PowerShell script.

(v) Errors detectable by PowerShell and concerning the email being sent. For example, the email address must have the final extension, e.g. ".it" or ".com". If no extension is included, PowerShell displays an error message.

The user can verify the presence of most of these errors in different ways. To verify if error (i) occurred, the user can rely on the `notification()` option. To verify if error (ii) occurred, nothing can be done—to the best of our knowledge. This error is equivalent to not plugging in a desktop computer and expecting it to display an error message; so, the user must be careful in inputting the server and port information. To verify if error (iii) occurred, the user must rely on Stata error messages. Errors type (iv) cannot occur—to the best of our knowledge, if the ado file that includes the PowerShell script is correct. To verify if errors type (v) occurred, the user can rely on the combination of `erroroption()`, `errormessage()`, and `pstimeout()`.

If the user is sending very many emails, it is possible to verify which emails have been commanded to be sent by inputting the information in an ad-hoc dataset in the folder with `.ps1` files. For that purpose, it is possible to use several commands. Here is an example of how to use `filelist`—a user-contributed package that should be downloaded—with this purpose:

```
ssc install filelist
cap erase .\SUCCESSemails.dta
filelist, dir(.\) pat("*.ps1") save(.\SUCCESSemails.dta) replace
```

This command grabs all the `.ps1` files with personalized and meaningful names—as suggested in Example 2—and inputs them in a dataset called `SUCCESSemails`.

The user can check individual `.ps1` files too; these files can be opened with Notepad and Notepad++, for example.

8 Conclusion

The **sendemails** package allows users to automatically send emails with Stata through PowerShell, which is open-source and cross-platform. Using this package researchers can perform various email tasks.

This package has one potential limitation. Protocols established by email providers are ever-changing, which could cause the future failure of **sendemails**. While `Send-MailMessage`, and thus this package, currently works with the most popular email providers, there is no alternative `cdmlet` that can be universally adapted to any email provider yet. However, there are at least two solutions. (A) The security procedure can be updated. For example, this situation has already occurred in between revisions of this manuscript, when at the end of May 2022 Google adopted a new third-party sign-in policy—we described the procedure to deal with this issue in Appendix 1. (B) Whenever a new universal `cdmlet` will be available, `Send-MailMessage` can be updated to let **sendemails** work with the new protocols.

9 Programs and supplemental materials

Please visit the website: https://sites.google.com/site/lucafumarco/stata-packages for the complete package material.

Acknowledgments

We thank an anonymous referee for the constructive suggestions and comments. We owe a debt of gratitude to Michaela Kecskésová for the research assistance. Luca Fumarco acknowledges the generous support from the NPO "Systemic Risk Institute" number LX22NPO5101, funded by European Union - Next Generation EU (Ministry of Education, Youth and Sports, NPO: EXCELES) and from the CERGE-EI Foundation Teaching Fellowship Program. We thank Clyde Schechter and Sam Langfield for their help in solving some package bugs.

References

Baert, Stijn. 2018. "Hiring Discrimination: An Overview of (almost) All Correspondence Experiments since 2005." In *Audit Studies: Behind the Scenes with Theory, Method, and Nuance*, edited by S. M. Gaddis, 63–77. Cham, Switzerland: Springer.

Button, Patrick, Eva Dils, Benjamin Harrell, Luca Fumarco, David Schwegman. 2020. "Gender Identity, Race, and Ethnicity Discrimination in Access to Mental Health Care: Preliminary Evidence from a Multi-Wave Audit Field Experiment." *NBER* Working Paper. Available at: https://www.nber.org/papers/w28164

Carlsson, Magnus, Dan-Olof Rooth. 2007. "Evidence of Ethnic Discrimination in the Swedish Labor Market Using Experimental Data." *Labour Economics* 14(4):716-29.

Chehras, Nanneh. 2017. Automating correspondence study applications with python and SQL: Guide and code. Mimeo.

Crabtree, Charles. 2018. "An Introduction to Conducting Email Audit Studies." In *Audit Studies: Behind the Scenes with Theory, Method, and Nuance*, edited by S. M. Gaddis, 103–117. Cham, Switzerland: Springer.

Fumarco, Luca. 2017. "Disability Discrimination in the Italian Rental Housing Market: A Field Experiment with Blind Tenants." *Land Economics* 93(4):567-84.

Gaddis, S. Michael. 2015. "Discrimination in the Credential Society: An Audit Study of Race and College Selectivity in the Labor Market." *Social Forces* 93(4):1451–79.

Gaddis, S. Michael. 2018a. "An Introduction to Audit Studies in the Social Sciences." In *Audit Studies: Behind the Scenes with Theory, Method, and Nuance*, edited by S. M. Gaddis, 3–44. Cham, Switzerland: Springer.

Gaddis, S. Michael. 2018b. *Audit Studies: Behind the Scenes with Theory, Method, and Nuance*. Cham, Switzerland: Springer.

Gaddis, S. M., DiRago, N. 2021. "Housing Audit Studies in the Social Sciences." *SSRN* Working Paper. Available at https://papers.ssrn.com/abstract=3796335

Gaddis, S. Michael, and Raj Ghoshal. 2015. "Arab American Housing Discrimination, Ethnic Competition, and the Contact Hypothesis." *Annals of the American Academy of Political and Social Science* 660(1):282–99.

Gaddis, S. Michael, and Raj Ghoshal. 2020. "Searching for a Roommate: A Correspondence Audit Examining Racial/Ethnic and Immigrant Discrimination among Millennials." *Socius* 6: 1-16.

Lahey, Joanna N., and Ryan A. Beasley. 2009. Computerizing audit studies. *Journal of Economic Behavior & Organization 70*(3): 508–514.

Neumark, David, Ian Burn, and Patrick Button. 2019. "Is It Harder for Older Workers to Find Jobs? New and Improved Evidence from a Field Experiment." *Journal of Political Economy*¸127(2):922-70.

Oh, Sun Jung, John Yinger. 2015. "What Have We Learned from Paired Testing in Housing Markets?" *Cityscape,* 17(3):15–60.

Quillian, Lincoln, Devah Pager, Ole Hexel, and Arnfinn Midtbøen. 2017. "The Persistence of Racial Discrimination: A Meta-analysis of Field Experiments in Hiring over Time." *Proceedings of the National Academy of Sciences,* 114(41):10870–5.

Rich, Judith. 2014. "What Do Field Experiments of Discrimination in Markets Tell Us? A Meta-analysis of Studies Conducted since 2000." *IZA* Working Paper, 8584. Available at

https://www.iza.org/publications/dp/8584/what-do-field-experiments-of-discrimination-in-markets-tell-us-a-meta-analysis-of-studies-conducted-since-2000.

Xuan Zhang & Dongliang Cui & Chuntao Li, 2013. "PSEMAIL: Stata module to send email from within Stata for Windows environment." *Statistical Software Components,* S457606, Boston College Department of Economics.

Zschirnt, Eva, and Didier Ruedin. 2016. "Ethnic Discrimination in Hiring Decisions: A Meta-analysis of Correspondence Tests 1990–2015." *Journal of Ethnic and Migration Studies,* 42(7):1115–34.

About the authors

Luca Fumarco is an Assistant Professor of Economics, at Masaryk University. His research interests include labor, education, and sports economics; he primarily investigates discrimination issues and relative age effects. He is also IZA Affiliate and GLO Fellow.

Michael Gaddis is an Associate Professor of Sociology, at University of California, Los Angeles. He is also a Faculty Fellow with the California Center for Population Research, and a Faculty Affiliate with the Center for Social Statistics. Finally, he is a Senior Research Scientist at Northwest Evaluation Association. His research interests include inequality, race and ethnicity, discrimination, labor markets, sociology of education, higher education, mental health and stigma, correspondence audits tests, and experimental methods.

Francesco Sarracino is a Senior Happiness Economist at the Research Division of STATEC—the National Institute of Statistics and Economic Studies of Luxembourg, and GLO fellow. His work

aims at identifying evidence-based policies to promote people's well-being and sustainable development; he also contributes to methodological survey research.

Iain Snoddy is Associate at Analysis Group. He completed his PhD in Economics from the Vancouver School of Economics at the University of British Columbia. His research focuses on analyzing the labor market and novel econometric methods.

Appendix 1

Table A.1 SMTP servers and ports.

Service	SMTP server	Port	Connection
Outlook.com	`smtp-mail.outlook.com`	587	TLS
Yahoo mail	`smtp.mail.yahoo.com`	587	TLS
Windows Live Hotmail	`smtp.live.com`	587	TLS
Zoho	`smtp.zoho.com`	587	TLS
Gmail	`smtp.gmail.com`	587	TLS

Setting up Gmail

On May 30th, 2022, Google adopted a new third-party sign-in policy, which does not allow users to log-in via a third-party app with a standard username and password combination. Since then, to use **`sendemails`** with Gmail accounts, the user needs to carry out two additional steps, which lead to the generation of an app-specific password, which substitutes the email password for **`sendemails`**. The actual password of Gmail is not changed. This password is an automatically generated 16-digit passcode, generated by Gmail itself. The user should carry out the following two steps:

A) Enabling the 2-step verification

The account needs to have the two-factor authentication enabled. This can be done by logging into the account, then going to Manage your Google account → Security → Sign in to Google → 2-Step Verification → add the user's phone number, choose a backup option and press SEND → type in the received code → TURN ON.

B) Generating app-specific password for Stata

The user should repeat the same procedure in A) up until "Sign in to Google," but now, instead of 2-Step Verification, the user should go to "App passwords." In the "Select app," the user should choose "Other (custom name)" and give a name to the app (e.g., sendemails) and click GENERATE. The user should now see an app-specific password that we advise to be carefully noted somewhere: after this window is closed, the token is gone. This token is the actual password that will be used by **`sendemails`**, so it should be written in the password txt-file, in place of the actual Gmail account password.

Appendix 2

`sendemails` for correspondence tests: a brief discussion and an example

`sendemails` can be very useful to conduct online experiments, such as correspondence audit tests. In these experiments, social scientists send emails from fictitious people over a long period. These emails are similar to each other, except for one aspect which reflects a characteristic of interest to the researcher. This method has been used to covertly detect discrimination in various markets based on race/ethnicity, gender, age, sexual orientation, gender identity, disability, and other characteristics (Button et al. 2020; Carlsson and Rooth 2007; Gaddis 2018a, 2018b; Fumarco 2017; Neumark, Burn, and Button 2019).[4] Often time, this type of experiment requires experimenters to send multiple attachments as well. To conduct these experiments, some researchers have created ad-hoc solutions (Lahey and Beasley 2009; Gaddis 2015; Gaddis and Ghoshal 2015, 2020). These solutions have been developed for Python, SQL, and R (Chehras 2017; Crabtree 2018). Alternative solutions are expensive, such as hiring

[4] Social scientists have increasingly used this experimental method in recent years (Baert 2018; Gaddis 2018a; Gaddis and DiRago 2021; Oh and Yinger 2015; Quillian et al. 2017; Rich 2014; Zschirnt and Rueding 2016).

freelancers (Crabtree 2018). `sendemails` — the first Stata package able to send emails from different accounts and with multiple attachments — provides a cheap and versatile tool that can come in handy with correspondence tests. Below, we provide an example of how to conduct a correspondence test with `sendemails`.

Let us assume that the experiment is conducted over two days only, and the first day of the experiment is April 17th, 2022.

For this example, the user does not have to load a dataset again, because Stata still has the dataset `"excelstatajexamples.dta"` in memory — partially used in Subsection 5.2. In this example, this dataset will be used in full.

The computer should be turned on, and Stata should be open whenever the experiment is run. Notice that emails are not created beforehand, saved on the server of the email provider, and sent at the right time automatically.

Day one of the experiment

Before executing the part of the do-file to conduct the actual experiment, the user should create a "date" variable. This can be done, for instance, as follows:

```
     //Generate date variable that we use for the loop:
     cap drop date_send
     gen date_send= "17apr2023"
     //TODAYDATE each day when you want to conduct the
experiment, you change this date and Stata will send only those
messages for which (see more below)
     replace date_send= "17apr2023" in 1
     replace date_send= "16apr2023" in 3
     replace date_send= "21apr2023" in 2

     cap drop datestand
     gen datestand = daily(date_send, "DMY")
     //Stata loops over the number of days since January 1, 1960
     format datestand %td
```

```
     //Tell Stata to create temporary information about the time
and date when the email has been sent:
     cap drop date_sent
     gen date_sent = ""
     cap drop time_sent
     gen time_sent = ""

     cap drop deliverydate
     cap drop deliverytime
     gen deliverydate = ""
     gen deliverytime = ""
```

What follows is the part of the do-file that would be run on each day the experiment is conducted.

```
     set more off

     //generate today date

     display "`c(current_date)'"
     cap drop today
     gen today=daily("`c(current_date)'", "DMY")
     format today %td

     cap drop tmp
     gen tmp= _n if datestand==today

     cap drop identifier
     egen identifier= rank(tmp)

     //Stata gives an increasing number to those emails for
which datestand==today, that is, those emails for which the
planned date for the email delivery equals today; this number
will be used to loop sendemails only over those days

     //Loop over the observations for which datestand==today:

     sum identifier if datestand==today
     forval i= `r(min)'/`r(max)' {

     // create temporary variables and locals that Stata uses to
populate the sendemails command. These values are based on the
information in the dataset

     cap drop `bodystr'
     tempvar bodystr
```

```
gene `bodystr'=_n if identifier==`i' & datestand==today
summ `bodystr', meanonly
local index=r(mean)
local b=body[`index']

cap drop `serverstr'
tempvar serverstr
gene `serverstr'=_n if identifier==`i' & datestand==today
summ `serverstr', meanonly
local index=r(mean)
local s=server[`index']

cap drop `portstr'
tempvar portstr
gene `portstr'=_n if identifier==`i' & datestand==today
summ `portstr', meanonly
local index=r(mean)
local po=port[`index']

cap drop `userstr'
tempvar userstr
gene `userstr'=_n if identifier==`i' & datestand==today
summ `userstr', meanonly
local index=r(mean)
local u=username[`index']

cap drop `passstr'
tempvar passstr
gene `passstr'=_n if identifier==`i' & datestand==today
summ `passstr', meanonly
local index=r(mean)
local pa=password[`index']

cap drop `friendstr'
tempvar friendstr
gene `friendstr'=_n if identifier==`i' & datestand==today
summ `friendstr', meanonly
local index=r(mean)
local r=receiver[`index']

cap drop `subjectstr'
tempvar subjectstr
gene `subjectstr'=_n if identifier==`i' & datestand==today
summ `subjectstr', meanonly
local index=r(mean)
local sbj=saluto[`index']
```

```
          sendemails `r', ///
     body(`b') para(1 2 3)  sub(`sbj')  ///
     smtpport(`po') smtpserver(`s') /// the user can try to omit
these two options, if you use outlook
     ufile(`u') ///
     pfile(`pa') ///
     psloc(.\)

     replace date_sent= "`c(current_date)'" if identifier==`i'
     replace time_sent= "`c(current_time)'" if identifier==`i'
     }

     replace deliverydate = date_sent if deliverydate== ""
     replace deliverytime = time_sent if deliverytime== ""

     cap drop __*

save "statajexamplescorrtests.dta", replace
```

On this first day, only the email in the fourth row is sent. The dataset is now called `"statajexamplescorrtests.dta"`.

Day two of the experiment

After the user has run the first day of the experiment, the second day could be run immediately, with the help of a little trick. This can be done by changing `date_send`, for example as follows:

```
     cap drop date_send
     gen date_send= "18apr2023"
     replace date_send= "21apr2023" in 4
     replace date_send= "17apr2023" in 3
     replace date_send= "17apr2023" in 2

     cap drop datestand
     gen datestand = daily(date_send, "DMY")
     format datestand %td
```

What follows is the part of the do-file that would be run on each day the experiment is conducted.

```
set more off
display "`c(current_date)'"
cap drop today
gen today=daily("`c(current_date)'", "DMY")
format today %td

cap drop tmp
gen tmp= _n if datestand==today

cap drop identifier
egen identifier= rank(tmp)

sum identifier if datestand==today
forval i= `r(min)'/`r(max)' {

cap drop `bodystr'
tempvar bodystr
gene `bodystr'=_n if identifier==`i' & datestand==today
summ `bodystr', meanonly
local index=r(mean)
local b=body[`index']

cap drop `serverstr'
tempvar serverstr
gene `serverstr'=_n if identifier==`i' & datestand==today
summ `serverstr', meanonly
local index=r(mean)
local s=server[`index']

cap drop `portstr'
tempvar portstr
gene `portstr'=_n if identifier==`i' & datestand==today
summ `portstr', meanonly
local index=r(mean)
local po=port[`index']

cap drop `userstr'
tempvar userstr
gene `userstr'=_n if identifier==`i' & datestand==today
summ `userstr', meanonly
local index=r(mean)
local u=username[`index']

cap drop `passstr'
tempvar passstr
gene `passstr'=_n if identifier==`i' & datestand==today
summ `passstr', meanonly
```

```
	local index=r(mean)
	local pa=password[`index']

	cap drop `friendstr'
	tempvar friendstr
	gene `friendstr'=_n if identifier==`i' & datestand==today
	summ `friendstr', meanonly
	local index=r(mean)
	local r=receiver[`index']

	cap drop `subjectstr'
	tempvar subjectstr
	gene `subjectstr'=_n if identifier==`i' & datestand==today
	summ `subjectstr', meanonly
	local index=r(mean)
	local sbj=saluto[`index']

	sendemails `r', ///
	body(`b') para(1 2 3)  sub(`sbj')  ///
	smtpport(`po') smtpserver(`s') /// the user can try to omit
these two options, if you use outlook
	ufile(`u') ///
	pfile(`pa') ///
	psloc(.\)

	replace date_sent="`c(current_date)'" if identifier==`i'
	replace time_sent="`c(current_time)' " if identifier==`i'
	}

	replace deliverydate = date_sent if deliverydate== ""
	replace deliverytime = time_sent if deliverytime== ""

	cap drop __*

	save "statajexamplescorrtests.dta", replace
```