PharmaSUG 2020 - Paper AD-308

Using Data-Driven Python to Automate and Monitor SAS® Jobs

Julie Stofel, SCHARP at Fred Hutchinson Cancer Research Center, Seattle, Washington

ABSTRACT

This paper describes how to integrate Python and SAS® to run, evaluate, and report on multiple SAS programs with a single Python script. It discusses running SAS programs in multiple environments (Latin-1 or UTF-8 encoding; command-line or cron submission) and ways to avoid potential Python version issues. A handy SAS-to-Python function guide is provided to help SAS programmers new to Python find the appropriate Python method for a variety of common tasks. Methods to find and search files, run SAS code, read SAS data sets and formats, return program status, and send formatted emails are demonstrated in Step-by-Step instructions. The full Python script is provided in the Appendix.

INTRODUCTION

The Statistical Center for HIV/AIDS Research and Prevention at the Fred Hutchinson Cancer Research Center (SCHARP) has 20 years of experience managing clinical trials data in SAS®. We are shifting the non-statistical parts of our work, particularly administrative tasks, from SAS to Python. This paper provides a sample Python program that uses data-driven programming to run, check, and report on a set of SAS programs across multiple studies. Tips and tricks are provided to help ensure the right version of SAS and the right version of Python are run in different environments. This paper is geared toward SAS users new to Python. Therefore, details are provided for all aspects of creating the Python script, with SAS programs referenced as examples. The Python script is written as a simple transactional script designed to be familiar to SAS users. This has a different look than modular Python scripts that the user may have seen in other sample Python code. The specifics described here are for a Linux-based SAS and Python computing environments, but these concepts and techniques apply to all computing environments.

WHICH PYTHON?

The first thing to know about Python is that it is an open source language with many different versions available for download, and a nearly infinite number of packages written by Python users across the world and made available for public use. This means that for any problem you have, there is probably already a package that solves that problem. Users can mix and match packages and versions and can take advantage of new or esoteric packages as they are developed. This also means that packages can become obsolete over time as the language evolves, and that the same package can have different behaviors depending on which version of the language is loaded. To solve this problem, we have a centrally managed production version of Python (currently 3.7) with a limited number of installed packages. We run production code in the controlled production environment to ensure reproducibility.

The first line of the Python script (known as the ‘shebang’) defines the version of Python to use:

```
#!/usr/local/apps/python/python3-controlled/bin/python
```

The shebang is invoked when the Python script is made executable:

```
chmod 775 myscript.py
```

and run as an executable:

```
./myscript.py
```

WHICH SAS?

We have studies encoded in both UTF-8 and Latin-1. Therefore, our administration script must be able to switch between encodings as appropriate. In addition, the script is run on different servers by different users (people and systems), so we must be able to specify the correct path in each environment.

The `which` system command shows the correct path for any given command. In our case, SAS Latin-1 encoding has the shortcut `sas` in our system, and SAS UTF-8 has the shortcut `sas_u8`,

so:

```
which sas
```

displays:

```
/usr/local/apps/bin/sas
```

and:

```
which sas_u8
```

displays:

```
/usr/local/apps/bin/sas_u8
```

In the Python script, these paths are explicitly set as variables that are used when invoking SAS from Python:

```
sas_cmd = "/usr/local/apps/bin/sas"
sas_u8_cmd = "/usr/local/apps/bin/sas_u8"
```

PYTHON PACKAGES

Unlike SAS, in which all capability is available whenever the program is invoked, Python loads with minimum capabilities. You must explicitly load all the packages that you will be using in the script.

SYSTEM PACKAGES

sys : provides access to variables used or maintained by the interpreter and to functions that interact strongly with the interpreter [1]. It is used in the example program to identify the version of Python being run, and to create a log file similar to a SAS log file.

os: provides a portable way of using operating system dependent functionality [2]. It is used in the example program to change directories and find files

subprocess: spawns new processes, connects to their input/output/error pipes, and obtain their return codes.[3] It is used in the example program to run SAS programs and obtain their return codes (error codes), as well as to use the `grep` command to search files and return results.

smtplib: defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.[4] This is used to send the summary email of the results.

logging: defines functions and classes which implement a flexible event logging system for applications and libraries.[5]

SPECIALIZED PACKAGES

Pandas: an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.[6] This is the primary package we use for data management and analysis. It can be used to read and manipulate SAS files, as well as other data types. It is used in this script to read and subset a SAS metadata

data set and a format data set (created from a format catalog) to determine which studies to run and to determine file names.

SASPy: a Python module written by SAS to provide Python APIs to the SAS system, enabling users to start a SAS session and run SAS code from Python.[7] It is used in this script to read a SAS format catalog, then use the `cntlout` option to write a SAS data set from the format procedure.

CONFIGURATION FILE FOR SASPy

SASPy requires a configuration file [8] in order to start a SAS session:

```
sas = saspy.SASsession(cfgfile=cfgfile)
```

A sample (template) configuration file can be found in:

```
<your python installation location>/site-packages/saspy/sascfg.py
```

For the example described in this paper, for example, our Python installation location (see WHICH PYTHON? above) is:

```
/usr/local/apps/python/python3-controlled/lib/python3.7
```

The programmer uses the template configuration file to creates their own sascfg_personal.py in their /home directory, where it is accessed by default by the SASPy module. However, rather than create a single personal sascfg file that includes multiple options and is accessed by default behavior, the programmer may find it simpler to create simple versions of the sascfg file that are called explicitly. While the template file has 256 lines (including comments explaining each option), only the following 8 lines are required to open a Latin-1 session:

```
SAS_config_names=['default']
SAS_config_options = {'lock_down': False,
                      'verbose'  : True
                     }
SAS_output_options = {'output' : 'html5'}
default  = {'saspath'  : '/usr/local/apps/bin/sas',
            'encoding': 'latin1'
            }
```

and similarly, the following 8 lines will open a UTF-8 session:

```
SAS_config_names=['default']
SAS_config_options = {'lock_down': False,
                      'verbose'  : True
                     }
SAS_output_options = {'output' : 'html5'}
default  = {'saspath'  : '/usr/local/apps/bin/sas_u8',
            'encoding': 'utf8'
            }
```

PYTHON TYPOGRAPHY

There are several important differences between how SAS and Python respond to how a script is created in a text-editor.

1. Unlike SAS, Python requires no line ending punctuation (;)
2. Unlike SAS but common to many other languages, Python is case-sensitive, so `myVarname` is not the same as `myvarname`.

3. Unique to Python: if/then, do/while, and other similar statements have the following required conventions:
    a. the “do” statement is replaced by a semi-colon (:)
    b. there is no “end” statement
    c. indentation is required to define the full statement

This is most easily seen in the following examples:

In SAS,

```
%for i in %sysfunc(countw(&mylist)) %then %do;
myfile = %scan(&mylist, &);
%if %sysfunc(fileexist(&myfile)) %then %do;
x “cp &myfile ../another_location”;
%end;
```

compiles the same as

```
%for i in %sysfunc(countw(&mylist)) %then %do;
      myfile = %scan(&mylist, &);
      %if %sysfunc(fileexist(&myfile)) %then %do;
            x “cp &myfile ../another_location”;
      %end;
%end;
```

The only difference in the above SAS examples is that the second is easier (for humans) to read, because the lines are indented according to the task to perform. However, SAS knows to perform the statements between the %do and %end tags, regardless of how those statements are indented.

In Python, in contrast, there are no end tags. The compiler relies on the exact number of indents to perform the task.

The following statement in Python

```
for i in mylist:
if os.path.exists(i):
cp i ../another_location/.
```

will not compile (will throw an error), while

```
for i in mylist:
      if os.path.exists(i):
            cp i ../another_location/.
```

will compile and run.

COMPARISON OF SAS AND PYTHON METHODS FOR COMMON TASKS

The Pandas data frame is similar to a SAS data set in that it has rows (records) and columns (variables). However, syntax for selecting, summarizing, and displaying data frame records may be unfamiliar to many SAS users, so a brief comparison of methods is presented here:

Task	SAS	Python
Select records from a data set called ‘all’ where the (case-insensitive) value of protstat is ’open’	**WHERE** data open; set all; where lowcase(protstat) = ‘open’; run;	**LOC** *Access a group of rows and columns by label(s)* open = all.loc[all.protstat.str.lower() == ‘open’]
Print the top 10 records	**OBS** proc print data = open(obs=10);	**HEAD()** *first 10 rows* print(all.head())

Task	SAS	Python
Does a file exist?	**FILEEXIST** %sysfunc(fileexist(myfile.sas))	**OS.PATH.EXISTS** os.path.exists(myfile.sas)
Loop through list / array	**DO** %do i = 1 %to %sysfunc(countw(&list)); <to do item> %end;	**FOR** for i in list: <to do item>

STEP-BY-STEP GUIDE

The following steps show how to write a Python program to perform a variety of useful tasks, including setting up the Python and SAS environments, search for files, open data sets and search for values in them, define variables, run programs, determine completion status, search logs for errors and warnings, and email a summary report of results.

STEP 1: DEFINE PYTHON VERSION TO USE

```
#!/usr/local/apps/python/python3-controlled/bin/python
```

STEP 2: START PYTHON LOG

```
import sys

#Create a function that writes status to log and terminal
class Tee(object):
    def __init__(self, *files):
        self.files = files
    def write(self, obj):
        for f in self.files:
            f.write(obj)
    def flush(self):
        pass
#Start the python log
f = open('test.logfile', 'w')
backup = sys.stdout
sys.stdout = Tee(sys.stdout, f)
```

STEP 3: GET CURRENT WORKING DIRECTORY AND CHECK PYTHON VERSION

```
#Get current working directory
cwdpath = os.getcwd()

#Check the Python version and paths you are running from
print("\n \n Running Python version " + str(sys.version_info.major) + "." + str(sys.version_info.minor) +
" in the following paths:")
for i in sys.path:
    print(i)
if sys.version_info.major != 3:
    print("\n \n This script must be run in Python 3. This is an executable script that runs in the correct
version of Python if run with the ./<script>.py command on command line, or with the full path in cron")
    print(" Exiting Python \n \n")
    exit()
```

STEP 4: IMPORT PYTHON MODULES

```
import os
import subprocess
import smtplib
import logging
import pandas as pd
import saspy
```

STEP 5: DEFINE SAS COMMANDS

```
#Latin-1
sas_cmd = "/usr/local/apps/bin/sas"

#UTF-8
sas_u8_cmd = "/usr/local/apps/bin/sas_u8"
```

STEP 6: CHECK AND CREATE SAS FORMAT CATALOG

In this step, we first check to see if the catalog (formats.sas7bcat) already exists. If it does not exist, we create it by running a SAS program.

```
#Check if file exists
if os.path.exists(cwdpath + "/formats.sas7bcat"):
    print("\n Reading format catalog from " + cwdpath + "/formats.sas7bcat")
else:
    print("\n Creating Standard SAS Format catalog in local directory " + cwdpath )
    cmd = sas_cmd + " /trials/sas/SASformats/formats.sas"

    #Run the command with the subprocess module
    run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
    #Get the results (return code) of the process
    run_cmd.communicate()
    #Check return code - if did not return with 0 return code, exit
    if run_cmd.returncode != 0:
        print("There was a problem running " + cmd + " Return code is " + str(run_cmd.returncode) )
        exit()
    else:
        print(cmd + " ran sucessfully" )
```

STEP 7: READ SAS FORMAT CATALOG

In this step, we read the format catalog with SAS format procedure and write it to a SAS data set

```
#Start the SAS session
cfgfile= cwdpath + "/sascfg_sas.py"
sas = saspy.SASsession(cfgfile=cfgfile)
sas.submit("libname loc '" + cwdpath +"';")
saslog=sas.submit('proc format lib=loc cntlout=loc.fmt; run;')
print(saslog)
#End the SAS session
sas._endsas()
```

STEP 8: READ SAS DATA SETS INTO PANDAS DATA FRAMES

Read data set with pandas.read_sas. In Python 3, you have to explicitly define encoding.

```
#Read in formats exported in the previous step
fmtfile=cwdpath + "/fmt.sas7bdat"
fmt = pd.read_sas(fmtfile, format="sas7bdat", encoding="latin-1")

#Read in list of studies from study-site metadata data set
df = pd.read_sas(r'/trials/hivnet/data/protocolsiteinfo.sas7bdat', format='sas7bdat', encoding="latin-1")
```

STEP 9: SELECT DESIRED RECORDS FROM DATA FRAME

```
#Select records where the variable protstat is 'open'
open_studies = df.loc[df.protstat.str.lower()=='open']

#Show the top 10 results
print(open_studies.head())

#Get list (array) of unique studies from column 0 (first column) in the data frame
prot_array = open_studies[open_studies.columns[0]].unique()

#Sort the list (array)
prot_array.sort()
```

STEP 10: START REPORT

```
html_string = "Please review the following report. Update verification code to rectify discrepancies and
resolve errors and warnings <br> ";
```

STEP 11: LOOP THROUGH RECORDS AND DEFINE VARIABLES

```
for prot in prot_array:
    #Get all information records for each study out of the format catalog
    prot_info = fmt.loc[fmt.START.str.lower()==prot
    #Get the study directory: the value of the first record (0) for the variable LABEL
    sdir = prot_info.loc[prot_info.FMTNAME=='SASDIR'].at[0, 'LABEL']
    #Get the enrollment data set name – note that not all studies have an enrollment data set
    enr_df= prot_info.loc[prot_info.FMTNAME=='TLA_TO_ENR']
    if (enr_df.size > 0):
        enrds = enr_df.at[0, 'LABEL']
    else:
        enrds = " "
```

STEP 12: RUN SAS PROGRAMS AND WRITE OUTPUT TO REPORT STRING

```
if enrds != "":
        html_string = html_string + "<br> <br> <b>" + prot + " Results: </b> "
        if os.path.exists("/trials/hivnet/data/"+enrds+".sas7bdat"):
            vdir = sdir+"/code/verification/"+enrds+"/code".strip()
            vcode = vdir+"/v_"+enrds+"_autocode.sas".strip()
            vlog = vdir+"/v_"+enrds+"_autocode.log".strip()
            if os.path.exists(vcode):
                #Output directories - odir is for Linux searching, todir is for clickable link in email
                odir = sdir+"/code/verification/"+enrds+"/output".strip()
                todir = odir.replace("/trials", "file:///T:")
                #Move to the verification directory for the study
                os.chdir(vdir)
                #Run verification code in SAS UTF-8
                cmd = sas_u8_cmd + " " + vcode;
                run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
                run_cmd.communicate()
                #Check return code - if did not return with 0 return code, try again with SAS Latin-1
                if run_cmd.returncode != 0:
                    cmd = sas_cmd + " " + vcode;
                    run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
                    run_cmd.communicate()
                    if run_cmd.returncode == 0:
                        print(cmd + " ran ok. Return code is " + str(run_cmd.returncode) )
                    else:
                        html_string = html_string + "<br>     There is a problem with " + cmd +
                        html_string = html_string + " Return code is " + str(run_cmd.returncode)
                        if (run_cmd.returncode == 1):
                            #Grep the log for warnings
                            cmd = "grep 'WARNING' " + vlog
                            #Run the grep and put the results in an object called grep_wrn
                            grep_wrn = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE,
                                stderr=subprocess.PIPE)
                            #Read the output from grep_wrn and puts it in memory
                            output, _ = grep_wrn.communicate()
                            if grep_wrn.returncode == 0:
                                #Decode the output to a string and write to email
                                wrn_str = output.decode("latin-1")
                                html_string = html_string + "<br> " + wrn_str.replace("WARNING", "<br>  ")
                        if (run_cmd.returncode > 1):
                            #Grep the log for errors
                            cmd = "grep 'ERROR' " + vlog
                            #Run the grep and put the results in an object called grep_err
                            grep_err = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE,
                              stderr=subprocess.PIPE)
                            #Read the output from grep_err and put it in memory
                            output, _ = grep_err.communicate()
                            if grep_err.returncode == 0:
                                #Decode the output to a string and write string to email
                                err_str = output.decode("latin-1")
                                html_string = html_string + "<br> " + err_str.replace("ERROR", "<br>  ")
                else:
                    print(cmd + " ran ok" )
```

STEP 13: SEND EMAIL STATUS REPORT

```
#Import packages
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

#Define parameters
to_address = "julie@scharp.org"
from_address = "julie@scharp.org"
SERVER = "localhost"

# Create message container - the correct MIME type is multipart/alternative.
msg = MIMEMultipart('alternative')
msg['Subject'] = "Enrollment Verification Summary"
msg['From'] = from_address
msg['To'] = to_address

# The body of the message (a plain-text and an HTML version).
text = html_string
html = "<html><head></head><body>"+html_string+"</body></html>"

# Record the MIME types of both parts - text/plain and text/html.
part1 = MIMEText(text, 'plain')
part2 = MIMEText(html, 'html')

# Attach parts into message container.
# According to RFC 2046, the last part of a multipart message, in this case
# the HTML message, is best and preferred.
msg.attach(part1)
msg.attach(part2)

# Send the message via local SMTP server.
s = smtplib.SMTP(SERVER)
# sendmail function takes 3 arguments: sender's address, recipient's address
# and message to send - here it is sent as one string.
s.sendmail(from_address, to_address, msg.as_string())
s.quit()
```

CONCLUSION

This paper provides a detailed introduction for the SAS programmer who wants to take advantage of Python functionality to perform a variety of useful tasks, including searching for files, opening data sets, finding values, defining variables, and running full SAS programs as well as individual SAS data steps. Determining completion status, searching logs for errors and warnings, and emailing summary reports of results are also detailed.

REFERENCES

[1] PYTHON 3.8.1 DOCUMENTATION : SYS — SYSTEM-SPECIFIC PARAMETERS AND FUNCTIONS
https://docs.python.org/3/library/sys.html

[2] PYTHON 3.8.1 DOCUMENTATION : OS — MISCELLANEOUS OPERATING SYSTEM INTERFACES
https://docs.python.org/3/library/os.html

[3] PYTHON 3.8.1 DOCUMENTATION : SUBPROCESS — SUBPROCESS MANAGEMENT
https://docs.python.org/3/library/subprocess.html

[4] PYTHON 3.8.1 DOCUMENTATION : SMTPLIB — SMTP PROTOCOL CLIENT
https://docs.python.org/3/library/smtplib.html

[5] PYTHON 3.8.1 DOCUMENTATION : LOGGING — LOGGING FACILTY FOR PYTHON
https://docs.python.org/3/library/logging.html

[6] PANDAS: PYTHON DATA ANALYSIS LIBRARY
https://pandas.pydata.org

[7] SASPY: PYTHON APIS TO THE SAS SYSTEM
HTTPS://SASSOFTWARE.GITHUB.IO/SASPY/

[8] SASPY: CONFIGURATION
HTTPS://SASSOFTWARE.GITHUB.IO/SASPY/INSTALL.HTML#CONFIGURATION

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Julie Stofel
Company: Statistical Center for HIV/AIDS Research & Prevention (SCHARP) at Fred Hutchinson Cancer Research Center, Seattle
Email: julie@scharp.org

APPENDIX : FULL PYTHON SCRIPT

```
#!/usr/local/apps/python/python3-controlled/bin/python
#############################################################################################
#PROGRAM    run_enrollment_verification.py
#PURPOSE    run SAS verification programs from python, and demonstrate some python tricks:
#           read sas, find files, grep files, return status, send formatted emails
#AUTHOR     J Stofel
#DATE       04JAN2020
#OUTPUT     summary email report that provides information on verification discrepancies and
#           programmatic issues that need to be resolved
#############################################################################################
#DESCRIPTION
#The purpose of this script is to work within existing structure to summarize the results of
#verification for multiple studies.
#The script obtains the list of active studies from a study metadata file, then searches for and
#runs SAS verification code for the standard enrollment data set.
#It then runs the SAS verification report code to obtain the verification results.
#The SAS verification report code produces a pdf report and, if there are discrepancies,
#variable-level discrepancy reports in excel. Links to these results are written to a summary
#email report.
#The python script checks program success each time a program is run.  Since some studies are
#encoded in UTF-8 and some studies are encoded in Latin-1,
#The python script checks for program success first under UTF-8 (the most common option), then
#runs again under latin-1 if the program did not run successfully under UTF-8. If the program
#does not run successfully under either encoding, then the relevant portions of the log (showing
#e-r-r-o-r-s or w-a-r-n-i-n-g-s) are written to the summary email report.
#############################################################################################
#####START PYTHON PROGRAM#####################################################################
#Import python system functions
import sys
#Create a function that records status statements (print statements) to a log while they are also
written to terminal
class Tee(object):
    def __init__(self, *files):
        self.files = files
    def write(self, obj):
        for f in self.files:
            f.write(obj)
    def flush(self):
        pass
#Start the log
f = open('/trials/python/logfiles/run_enrollment_verification.logfile', 'w')
backup = sys.stdout
sys.stdout = Tee(sys.stdout, f)
#Check the python version and paths you are running from
print("\n \n Running python version " + str(sys.version_info.major) + "." +
str(sys.version_info.minor) + " in the following paths:")
for i in sys.path:
    print(i)
if sys.version_info.major != 3:
    print("\n \n This script must be run in python 3. This is an executable script that runs in
the correct version of python if run with the ./driver.py command on commandline, or full path in
cron")
    print(" Exiting python \n \n")
    exit()

#Define and check the paths to the SAS installations we will use
#Find these paths using the 'which sas' and 'which sas_u8' commands on Linux
#Latin-1
sas_cmd = "/usr/local/apps/bin/sas"
#UTF-8
sas_u8_cmd = "/usr/local/apps/bin/sas_u8"

#Import system and specialized packages used in program
import os                #operating system functions
import subprocess        #more sophisticated operating system functions
import smtplib           #emailing functions
import logging           #logging functions
import pandas as pd      #for data read/write/process
import saspy             #for running SAS programs
```

```
#Show environment
print("\nPython environment paths \n");
print(os.getenv('PATH'))
print("\nPath of current working directory")
cwdpath = os.getcwd()
print(cwdpath)

#Start the Program…
#Look for SAS format catalog - create it if it does not exist
if os.path.exists(cwdpath + "/formats.sas7bcat"):
    print("\n Reading format catalog from " + cwdpath + "/formats.sas7bcat")
else:
    print("\n Creating Standard SAS Format catalog in local directory " + cwdpath )
    cmd = sas_cmd + " /trials/sas/SASformats/formats.sas"
    print("Running " + cmd)
    run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
    run_cmd.communicate()
    #Check return code - if did not return with 0 return code, exit
    if run_cmd.returncode != 0:
        print("Problem running " + cmd + " Return code is " + str(run_cmd.returncode))
        exit()
    else:
        print(cmd + " ran sucessfully" )

#Export SAS Format Catalog into a SAS Data set. Use SASpy export catalog, pandas to read data set
print("\n Starting SAS session ")
cfgfile= "/trials/python/sascfg/sascfg_sas.py"
sas = saspy.SASsession(cfgfile=cfgfile)
sas.submit("libname loc '" + cwdpath +"';")
saslog=sas.submit('proc format lib=loc cntlout=loc.fmt; run;')
print("\n Results of sas call to read format catalog into data frame")
print(saslog)
print("\n Ending the SAS session")
sas._endsas()

#Read formats data set into dataframe with pandas.read_sas.
#In python 3, you have to explicitly define encoding
fmtfile = cwdpath + "/fmt.sas7bdat"
fmt = pd.read_sas(fmtfile, format = 'sas7bdat', encoding="latin-1")

#Read protocol_site_info sas data set with pandas.read_sas. This file has study status.
df = pd.read_sas(r'/trials/hivnet/data/protocol_site_info.sas7bdat',
                 format = 'sas7bdat', encoding="latin-1")

#Select studies from protocol_site_info where protocol status is 'Open'
open_studies = df.loc[df.protstat.str.lower()=='open']

#Get study acronyms from table of open studies: select column 0 (first column in dataframe)
#and make it unique. the result is an array
prot_array = open_studies[open_studies.columns[0]].unique()

#Sort using numpy sort (note - this cannot be combined with the line above)
prot_array.sort()

#Intialize message strings for output email
html_string = "Please review the following report. Update verification code to rectify " \
              + "discrepancies and resolve errors and warnings <br> "

#Now process each study: look for enrollment data set and verification code, and run the code
for prot in prot_array:
    print(prot)
    prot_info = fmt.loc[fmt.START.str.lower()==prot]
    print("\n Study Directory")
    prot_dir_df = prot_info.loc[prot_info.FMTNAME=='SASDIR']
    sdir = prot_dir_df.get_value(0, 3, takeable=True)
    print("Study dir for " + prot + " is " + sdir)
    print("\n Enrollment Data set")
    prot_enr_df = prot_info.loc[prot_info.FMTNAME=='TLA_TO_ENR']
    if (prot_enr_df.size > 0):
        enrds = prot_enr_df.get_value(0, 3, takeable=True)
```

```
else:
    enrds = ""
print("Enrollment data set for " + prot + " in TLA_TO_ENR format is " + enrds)
#If there is an enrollment data set recorded in the TLA_TO_ENR format, look for the data set
#Otherwise this is not a study of interest -- no status is written to email
if enrds != "":
    html_string = html_string + "<br> <br> <b>" + prot + " Results: </b> "
    if os.path.exists("/trials/hivnet/data/"+enrds+".sas7bdat"):
        print("\n The enrollment data set " + enrds + " exists")
        #Look for file sdir/code/verification/enrds/code/v_enrds_autocode.sas and run it
        vdir = sdir+"/code/verification/"+enrds+"/code".strip()
        vcode = vdir+"/v_"+enrds+"_autocode.sas".strip()
        vlog = vdir+"/v_"+enrds+"_autocode.log".strip()
        if os.path.exists(vcode):
            print("\n Found " + vcode)
            #Output strings: odir for Linux searching, todir for clickable link in email
            odir = sdir+"/code/verification/"+enrds+"/output".strip()
            todir = odir.replace("/trials", "file:///T:")
            print("Starting processing for " + vcode + " " + vdir + " \n")
            #Move to the verification directory for the study
            cmd = "cd " + vdir;
            print(cmd)
            #Move to local verification directory
            os.chdir(vdir)
            cwd = os.getcwd()
            print("You are now in " + cwd)
            #Run verification code in SAS UTF-8
            cmd = sas_u8_cmd + " " + vcode;
            print("Running " + cmd)
            run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
            run_cmd.communicate()

            #Check return code - if error (not 0 return code), try again with SAS Latin-1
            if run_cmd.returncode != 0:
                print("Problem running " + cmd + " Return code: " + str(run_cmd.returncode))
                cmd = sas_cmd + " " + vcode;
                print("Trying again with " + cmd)
                run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
                run_cmd.communicate()
                if run_cmd.returncode == 0:
                    print(cmd + " ran ok. Return code is " + str(run_cmd.returncode) )
                else:
                    print("Problem with " + cmd + " Return code:" + str(run_cmd.returncode))
                    html_string = html_string + "<br>     There is a problem with " \
                                + cmd + " Return code is " + str(run_cmd.returncode)
                    if run_cmd.returncode == 1:
                    #Grep the log for WARNING
                    cmd = "grep 'WARNING' " + vlog
                    grep_wrn = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE)
                    output, _ = grep_wrn.communicate()
                    if grep_wrn.returncode == 0:
                        #Decode the output to a string
                        wrn_str = output.decode("latin-1")
                        print(wrn_str)
                        #Write the string to the email
                        html_string = html_string + "<br> " + wrn_str.replace("WARNING",
                                      "<br>    ")
                    if run_cmd.returncode > 1:
                        #Grep the log for ERROR
                        cmd = "grep 'ERROR' " + vlog
                        #Run the grep and put the results in a Popen object called grep_err
                        grep_err = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE,
                                   stderr=subprocess.PIPE)
                        #Read the output from grep_err and put it in memory
                        #The output includes the return code (error code) and
                        #the command output, in this case the result of the grep
                        output, _ = grep_err.communicate()
                        if grep_err.returncode == 0:
                            #Decode the output to a string
                            err_str = output.decode("latin-1")
```

```
                print(err_str)
                #Write the string to the email
                html_string = html_string + "<br> " + err_str.replace("ERROR",
                            "<br>    ")
else:
    print(cmd + " ran ok" )

#Look for the compare code, and run it as well
if os.path.exists("compare.sas"):
    cmd = sas_u8_cmd + " compare.sas";
    print("Running " + cmd)
    run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
    run_cmd.communicate()
    #Check return code.
    #For this code, I know it runs with warnings, so accept 0 and 1 return codes
    if (run_cmd.returncode == 0 or run_cmd.returncode == 1):
        print(cmd + " ran ok")
    #If there is a problem, re-run in Latin-1
    else:
        print("There is a problem running " + cmd + " Return code is " + \
               str(run_cmd.returncode) )
        cmd = sas_cmd + " compare.sas";
        run_cmd = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
        run_cmd.communicate()
        if (run_cmd.returncode == 0 or run_cmd.returncode == 1):
            print(cmd + " ran ok" )
        else:
            print("There's also a problem with " + cmd + " Return code is " \
                     + str(run_cmd.returncode) )
            html_string = html_string + "<br>     Problem with " + cmd \
                     + " Return code is " + str(run_cmd.returncode)

    #Get the name of the compare report, with the date stamp
    #Note: the filename includes results, so it changes with circumstance.
    #But it always starts with the same string and ends in *.pdf...
    cmd = "ls -lt "+ odir + "/compare_stnddata.enr*.pdf"
    show_report = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
    output, _ = show_report.communicate()
    report_str = ""
    report_str = output.decode("latin-1").strip()
    if report_str == "":
        print("\n Output report: There is no report for compare_stnddata.enr*")
        print("\n Check name of the output -- is it not standardized?"
        html_string = html_string + "\n <br>   There is no report for " \
              + "compare_stnddata.enr* Check if report not standardized. <br>"
    else:
        print("\n Output report: "+report_str)
        #Get just the report name (without the date stamp)
        sloc = report_str.index('compare')
        pdf_report = report_str[(sloc):]
        #Add the report string to the message string
        html_string = html_string + "\n <br>   <a href='" + todir + "/" + \
               pdf_report + "'>" + report_str + "</a> <br>"
        #Print the name(s) of the xlsx file(s) to the terminal
        #Add names(s) to output string.
        #All files are of the form *VALUE_DIFFERENCES.xlsx
        cmd = "ls -lt ../output/*VALUE_DIFFERENCES.xlsx"
        #Get a string
        show_report = subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)
        output, _ = show_report.communicate()
        output_str = output.decode("latin-1").strip()
        #Insert indent at every newline....
        report_str = output_str.replace("\n", "\n  ")
        print("Adding " + report_str + " \n to message string and html string ")
        info_str = "Detail listings available for " \
                    + "variables with value differences " \
                    + " unless all values missing from one data set"
        html_string = html_string + "<i>  " + info_str + "</i> <br> "
        html_str = output_str.replace("-rw-rw-r--",
                                      "<br>     -rw-rw-r--")
        html_string = html_string + html_str
```

```
            else:
                print("There is no verification code " + vcode + " to run \n")
                html_string = html_string + " There is no verification code " + vcode + " to run"
        else:
            print("\n The enrollment data set " + enrds + " does not exist"
            html_string = html_string + " The enrollment data set /trials/hivnet/data/" \
                      + enrds + " does not exist"

#Send the email
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
to_address = "julie@scharp.org"
from_address = "julie@scharp.org"
SERVER = "localhost"
print("Sending email to " + to_address + " on " + SERVER \
       + ". The from address is " + from_address)

# Create message container - the correct MIME type is multipart/alternative.
msg = MIMEMultipart('alternative')
msg['Subject'] = "Enrollment Verification Summary"
msg['From'] = from_address
msg['To'] = to_address

# The body of the message (a plain-text and an HTML version).
text = html_string
html = "<html><head></head><body>"+html_string+"</body></html>"

# Record the MIME types of both parts - text/plain and text/html.
part1 = MIMEText(text, 'plain')
part2 = MIMEText(html, 'html')

# Attach parts into message container.
# According to RFC 2046, the last part of a multipart message, in this case
# the HTML message, is best and preferred.
msg.attach(part1)
msg.attach(part2)

# Send the message via local SMTP server.
s = smtplib.SMTP(SERVER)

# sendmail function takes 3 arguments: sender's address, recipient's address
# and message to send - here it is sent as one string.
s.sendmail(from_address, to_address, msg.as_string())
s.quit()

#Cleanup files created when obtaining formats
print("Moving back to " + cwd)
os.chdir(cwd)
cmd = "rm "+cwd+"/*format* "
subprocess.Popen(cmd, shell=True,stdout=subprocess.PIPE)

exit()
```