Everything is Better with Friends

Using SAS in Python Applications with SASPy and Open-Source Tooling (Beyond the Basics)

Setup for Part 4

Getting setup to use Google Colab with SAS OnDemand for Academics (ODA)

1. To execute code cells, you'll need credentials for the following accounts:
    - Google. (If you're not already signed in, you should see a **Sign In** button in the upper right corner. You can also visit https://accounts.google.com/signup to create an account for free.)
    - SAS OnDemand for Academics. (You can create an account for free at https://welcome.oda.sas.com/ using an existing SAS Profile account. If you don't already have a SAS Profile account, you can create one for free using the "Don't have a SAS Profile?" link on the ODA login page.)
2. We recommend enabling line numbers using the Tools menu: **Tools** -> **Settings** -> **Editor** -> **Show line numbers** -> **Save**
3. We also recommend enabling the Table of Contents using the View menu: **View** -> **Table of contents**
4. To save a copy of this notebook, along with any edits you make, please use the File menu: **File** -> **Save a copy in Drive**
5. Looking for "extra credit"? Please let us know if you spot any typos!

Connect to SAS OnDemand for Academics (ODA) and start a SAS session

Instructions:

1. Determine the Region for your ODA account by logging into https://welcome.oda.sas.com/. You should see the value `Asia Pacific`, `Europe`, or `United States` next to your username in the upper-right corner. (For more information about Regions, please see the ODA documentation.)
2. If your ODA account is associated with a Region other than `United States`, comment out Line 11 by adding a number sign (`#`) at the beginning of the line, and then do the following:
    - If your ODA account is associated with the Region `Europe`, uncomment Line 14 by removing the number sign (`#`) at the beginning of the line.
    - If your ODA account is associated with the Region `Asia Pacific`, uncomment Line 17 by removing the number sign (`#`) at the beginning of the line.
3. Click anywhere in the code cell, and run the cell using Shift-Enter.
4. At the prompt `Please enter the IOM user id`, enter either your SAS ODA user ID or the email address associated with your ODA account.
5. At the prompt `Please enter the password for IOM user`, enter the password for your SAS ODA account.

```
1  !pip install saspy
2
3  import saspy
4
5  sas = saspy.SASsession(
6      java='/usr/bin/java',
7      iomport=8591,
8      encoding='utf-8',
9
10     # The following line should be uncommented if, and only if, your ODA account is associated with the Region
11     iomhost = ['odaws01-usw2.oda.sas.com', 'odaws02-usw2.oda.sas.com', 'odaws03-usw2.oda.sas.com', 'odaws04-usw
12
13     # The following line should be uncommented if, and only if, your ODA account is associated with the Region
14     #iomhost = ['odaws01-euw1.oda.sas.com','odaws02-euw1.oda.sas.com'],
15
16     # The following line should be uncommented if, and only if, your ODA account is associated with the Region
17     #iomhost = ['odaws01-apse1.oda.sas.com','odaws02-apse1.oda.sas.com'],
18
```

```
)
print(sas)
```

```
Collecting saspy
  Downloading saspy-4.3.0.tar.gz (9.9 MB)
     |████████████████████████████████| 9.9 MB 16.7 MB/s
Building wheels for collected packages: saspy
  Building wheel for saspy (setup.py) ... done
  Created wheel for saspy: filename=saspy-4.3.0-py3-none-any.whl size=9929656 sha256=f11af7500c9bbcef35fdfc2
  Stored in directory: /root/.cache/pip/wheels/c3/b5/08/62c85da319a5178d19559f996ceefd7583b9bf31feeafbad8e
Successfully built saspy
Installing collected packages: saspy
Successfully installed saspy-4.3.0
Using SAS Config named: default
Please enter the IOM user id: isaiah.lankham@ucop.edu
Please enter the password for IOM user : ··········
SAS Connection established. Subprocess id is 136

Access Method         = IOM
SAS Config name       = default
SAS Config file       = /usr/local/lib/python3.7/dist-packages/saspy/sascfg.py
WORK Path             = /saswork/SAS_work4A2D0000D3B9_odaws04-usw2.oda.sas.com/SAS_work4FFC0000D3B9_odaws04-
SAS Version           = 9.04.01M6P11072018
SASPy Version         = 4.3.0
Teach me SAS          = False
Batch                 = False
Results               = Pandas
SAS Session Encoding  = utf-8
Python Encoding value = utf-8
SAS process Pid value = 54201
```

Note: This establishes a connection from Python in Google Colab to a SAS session running in SAS ODA.

Import additional packages

```
```

```
# We'll need several different features provided by the Flask web framework.
from flask import Flask, render_template, request

# We'll use the pathlib module to define Path objects pointing to local files.
from pathlib import Path
```

Part 4. Using Python to build simple web apps with SAS analytics

Section 4.1. Set Access Method: `localtunnel` or `ngrok`

Instructions:

- In the cell below, please set the variable `access_method` to be either `localtunnel` or `ngrok`. This will determine the behavior used in several cells below.
- As background, http://localtunnel.me/ is a free service used by many web developers. It doesn't require an access token, and it automatically provides HTTPS connections. However, because it's not a commerical product, it can be less reliable.
- On the other hand, `ngrok` is a freemium service requiring an access token. To create an access token, sign up at https://dashboard.ngrok.com/signup, and paste the token associated with your account below.

```
access_method = 'localtunnel'
# access_method = 'ngrok'

if access_method == 'localtunnel':

  # Install the localtunnel (reverse proxy) node package, which makes it possible to access locally
  # run web apps over the public Internet using the free http://localtunnel.me/ service.
  !npm install -g localtunnel

  # We'll also need a few, final modules to make a shell call that runs localtunnel in a
  # background thread at the same time we stand up a Flask web app.
  import os
  from time import sleep
```

```
  import threading

  # Define a function that starts the node package localtunnel and prints the resulting URL.
  def start_localtunnel_and_get_url():
      sleep(1)
      os.system("lt --port 5000 >> urls.txt 2>&1 &")
      sleep(1)
      print(Path('urls.txt').read_text().split('\n')[-2])

elif access_method == 'ngrok':
  # Install the ngrok (reverse proxy) plug-in for Flask, which makes it possible to access locally
  # run web apps over the public Internet using the https://ngrok.com/ service.
  !pip install flask-ngrok
  from flask_ngrok import run_with_ngrok

  # Configure public access token for ngrok.
  !pip install pyngrok==4.1.1
  !ngrok authtoken REPLACE_THIS_LONG_VARIABLE_NAME_WITH_YOUR_NGROK_ACCESS_TOKEN
```

```
/tools/node/bin/lt -> /tools/node/lib/node_modules/localtunnel/bin/lt.js
+ localtunnel@2.0.2
added 22 packages from 22 contributors in 1.79s
```

Concept Check 4.1

- Short Answer: List some ways to create a user-defined function in SAS.
- Fun Fact: In general, the use of `os.system` is frowned upon because it allows a Python application to execute arbitrary shell commands. However, since we're running this example inside of a Google Colab Sandbox, `os.system` is unlikely to cause any issue here.

Solution: Options include PROC FCMP, SCL (SAS Component Language), SAS/IML (Interactive Matrix Language), and the SAS Macro Facility.

Section 4.2. Run "Hello, World!" web app

```
# Define a Flask web app.
hello_work_web_app = Flask(__name__)

# Register a handler for an HTTP route for our web app.
@hello_work_web_app.route('/', methods=['GET'])
def handle_root_get_request():

    return 'Hello, World!'

# Run the web app, and look for a loca.lt or ngrok.io URL in the resulting output; visiting this
# URL will allow anyone with an Internet connection to interact with the app.
if access_method == 'localtunnel':
  threading.Thread(target=start_localtunnel_and_get_url).start()
elif access_method == 'ngrok':
  run_with_ngrok(hello_work_web_app)
hello_work_web_app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
your url is: https://true-games-wait-35-245-134-204.loca.lt
```

Concept Check 4.2

- Try this, and see what happens: Change the text displayed by the Flask web app.
- True or False: The example above can be modified to include multiple lines of text in the output of the Flask web app.
- Fun Fact/Hint: Python strings can be defined with several different quoting styles:
    - single quotes, as in `'Hello, World!'`
    - double quotes, as in `"Hello, World!"`
    - triple quotes, as in `'''Hello, World!'''` or `"""Hello, World!"""`

These styles are all interchangeable for single-line strings. However, unlike single- and double-quoted strings, triple-quoted strings can contain embedded line breaks.

Solution: True! We co either include embedded line breaks with the control sequence `\n`, like this:

```
return 'Hello,\nWorld!"
```

Or we can use a triple-coded string with embedded line breaks, like this:

```
return '''Hello,
World!'''
```

Section 4.3. Run web app with embedded SAS output

```
# Define a Flask web app.
web_app_with_embedded_sas_output = Flask(__name__)

# Register a handler for an HTTP route for our web app.
@web_app_with_embedded_sas_output.route('/', methods=['GET'])
def handle_root_get_request():

    sas_submit_results = sas.submit(
        '''
            proc print data=sashelp.class(obs=10); run;
        ''',
        results="HTML"
    )
    return sas_submit_results['LST']

# Run the web app, and look for a loca.lt or ngrok.io URL in the resulting output; visiting this
```

```
# URL will allow anyone with an Internet connection to interact with the app.
if access_method == 'localtunnel':
  threading.Thread(target=start_localtunnel_and_get_url).start()
elif access_method == 'ngrok':
  run_with_ngrok(web_app_with_embedded_sas_output)
web_app_with_embedded_sas_output.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
your url is: https://brave-eels-lose-35-245-134-204.loca.lt
```

Concept Check 4.3

- Try this, and see what happens: Change the SAS output displayed by the web app.
- True or False: The example above can be modified to include arbitrary SAS output.
- Fun Fact/Hint: The `submit` method can be used to pass arbitrary SAS code directly to a SAS kernel. After the SAS kernel executes the code, a dictionary (called `sas_submit_results` above) is returned with the following two key-value pairs:
  - `sas_submit_results['LST']` is a string comprising the results of executing the SAS code.
  - `sas_submit_results['LOG']` is a string comprising the plain-text log resulting from executing the SAS code.

Solution: True! As long as `sas_submit_results['LST']` isn't empty, we can change Line 10 to any SAS code executable by SAS ODA.

In addition, we could change Line 14 to `sas_submit_results['LOG']` in order to print a SAS log, instead.

Section 4.4. Additional Exercises

For practice, we recommend the following:

- Work through the Google Colab Notebook for the Dataset Explorer application.

Dataset Explorer is a proof-of-concept Flask web application that incorporates SAS analytics applied to user-selected datasets available in SAS ODA.

Notes and Resources

Want some ideas for what to do next? Here are our suggestions:

1. Continue learning Python.
    - For general programming, we recommend starting with these:
        - Automate the Boring Stuff with Python, a free online book with numerous beginner-friendly hands-on projects
        - Fluent Python, which provided a deep dive into Intermediate to Advanced Python concepts
    - For data science, we recommend starting with these:
        - A Whirlwind Tour of Python, a free online book with coverage of essential Python features commonly used in data science projects
        - The Unexpected Effectiveness of Python in Science, a PyCon 2017 keynote about the mosaic of vastly different use case for Python by the author of the *A Whirlwind Tour of Python*
        - Python for Data Analysis, which provided a deep dive into the `pandas` package by its creator, Wes McKinney
    - For web development in Python, we recommend starting with this:
        - The Flask Mega-Tutorial, a freely accessible series of blog posts covering essential features of developing dynamic websites with the `flask` web framework, including how to host them for free using a service called Heroku
2. Try using SASPy outside of Google Colab. For example, if you're interested in using a local SASPy environment, with Python talking to a commercial SAS installation, you're welcome to follow the setup instructions for the demo application

https://github.com/saspy-bffs/dataset-explorer

3. Try using SASPy outside of Google Colab. For example, if you're interested in using a local SASPy environment, with Python talking to a commercial SAS installation, you're welcome to follow the setup instructions for the demo application https://github.com/saspy-bffs/dataset-explorer
4. Keep in touch for follow-up questions/discussion (one of our favorite parts of teaching!) using isaiah.lankham@gmail.com and matthew.t.slaughter@gmail.com
5. If you have a GitHub account (or don't mind creating one), you can also chat with us on Gitter at https://gitter.im/saspy-bffs/community

In addition, you might also find the following documentation useful:

1. For more about the `flask` package, see https://flask.palletsprojects.com/
2. For more about the `flask-ngrok` package, see https://github.com/gstaff/flask-ngrok
3. For more about the `os` package, see https://docs.python.org/3/library/pathlib.html
4. For more about the `pathlib` package, see https://docs.python.org/3/library/pathlib.html
5. For more about the `pyngrok` package, see https://pyngrok.readthedocs.io/
6. For more about the `threading` package, see https://docs.python.org/3/library/pathlib.html
7. For more about the `time` package, see https://docs.python.org/3/library/pathlib.html
8. For more about the `saspy` package, including the methods used above, see the following:
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.submit
9. For more about some of the Python features used, such as functions, list indexing, and control flow with if-then-else conditionals, we recomend the following chapters of A Whirlwind Tour of Python:
    - https://jakevdp.github.io/WhirlwindTourOfPython/06-built-in-data-structures.html
    - https://jakevdp.github.io/WhirlwindTourOfPython/07-control-flow-statements.html
    - https://jakevdp.github.io/WhirlwindTourOfPython/08-defining-functions.html

10. For background on the HTTP Request/Response Cycle, we recommend the following:
    - Brief Overview: https://backend.turing.edu/module2/lessons/how_the_web_works_http
    - Deeper Overview: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
    - Summary of HTTP Status Codes: https://httpstatuses.com/
    - Google's Implementation of HTTP Status Code 418: https://www.google.com/teapot