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. Please enable line numbers using the Tools menu: **Tools** -> **Settings** -> **Editor** -> **Show line numbers** -> **Save**
2. To execute code examples, 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.)
3. To save a copy of this notebook, along with any edits you make, please use the File menu: **File** -> **Save a copy in Drive**
4. We also recommend enabling the Table of Contents using the View menu: **View** -> **Table of contents**
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 i
11     iomhost = ['odaws01-usw2.oda.sas.com', 'odaws02-usw2.oda.sas.com', 'odaws03-us
12 
13     # The following line should be uncommented if, and only if, your ODA account i
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 i
17     #iomhost = ['odaws01-apse1.oda.sas.com','odaws02-apse1.oda.sas.com'],
18 
19 )
20 print(sas)
```

```
Collecting saspy
  Downloading saspy-3.7.6.tar.gz (7.8 MB)
     |████████████████████████████████| 7.8 MB 3.1 MB/s
Building wheels for collected packages: saspy
  Building wheel for saspy (setup.py) ... done
  Created wheel for saspy: filename=saspy-3.7.6-py3-none-any.whl size=7858616 sh
  Stored in directory: /root/.cache/pip/wheels/b7/3c/56/3cec00a83001d0ffb2293f09
Successfully built saspy
Installing collected packages: saspy
Successfully installed saspy-3.7.6
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 126

Access Method         = IOM
SAS Config name       = default
SAS Config file       = /usr/local/lib/python3.7/dist-packages/saspy/sascfg.py
WORK Path             = /saswork/SAS_work88750000111B_odaws03-usw2.oda.sas.com/S
SAS Version           = 9.04.01M6P11072018
SASPy Version         = 3.7.6
Teach me SAS          = False
Batch                 = False
Results               = Pandas
SAS Session Encoding  = utf-8
Python Encoding value = utf-8
SAS process Pid value = 4379
```

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

Install and import additional packages

```
# Install the ngrok (reverse proxy) plug-in for flask, which makes it possible to
# web apps over the public Internet using the https://ngrok.com/ service
!pip install flask-ngrok

# Install the rich module for colorful printing
!pip install rich

# We'll use the datatime package to get the current datetime
from datetime import datetime

# We'll need several different features provided by the flask web framework and it
from flask import Flask, render_template, request
from flask_ngrok import run_with_ngrok

# We'll need the json package to turn string values from URLs into lists
import json

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

# We'll use the requests package to make a local copy of a file in the cloud
import requests

```

```
# We're overwriting the default print function with rich.print
from rich import print
```

```
Collecting flask-ngrok
  Downloading flask_ngrok-0.0.25-py3-none-any.whl (3.1 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/d
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/d
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/l
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/di
Installing collected packages: flask-ngrok
Successfully installed flask-ngrok-0.0.25
Collecting rich
  Downloading rich-10.15.2-py3-none-any.whl (214 kB)
     |████████████████████████████████| 214 kB 5.0 MB/s
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in /usr/local/lib/python3.
Collecting colorama<0.5.0,>=0.4.0
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Requirement already satisfied: typing-extensions<5.0,>=3.7.4 in /usr/local/lib/p
Collecting commonmark<0.10.0,>=0.9.0
  Downloading commonmark-0.9.1-py2.py3-none-any.whl (51 kB)
     |████████████████████████████████| 51 kB 6.0 MB/s
Installing collected packages: commonmark, colorama, rich
Successfully installed colorama-0.4.4 commonmark-0.9.1 rich-10.15.2
```

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

Section 4.1. Create sas_output_dir and template_dir

```
# Since we're going to build a web application, we first need to set up a couple o
# where we can store files.

# First, we'll use the current working directory (aka '.', per typical Unix file s
# for template files.
template_dir = Path('.')

# Next, we'll create a new directory called sas_output, where we'll store SAS resu
# run time from our SAS ODA session.
sas_output_dir = Path('sas_output')
sas_output_dir.mkdir(exist_ok=True)

# Finally, we can print the absolute paths of both directories.
```

```
14 print(f'The absolute path of template_dir is {template_dir.resolve()}')
15 print(f'The absolute path of sas_output_dir is {sas_output_dir.resolve()}')
```

```
The absolute path of template_dir is /content
The absolute path of sas_output_dir is /content/sas_output
```

Concept Check 4.1

1. True or False: Deleting the option `exist_ok=True` on Line 11 won't affect the creation of `sas_output_dir`.
2. True or False: In Python, the values `True` and `False` are interchangeable with the integer values `1` and `0`, respectively.

1. *It depends. The first time this cell is run, it's unlikely a directory called `'sas_output'` already exists. However, this directory does already exist, then removing `exist_ok=True` will cause a `FileExistsError`.*
2. *True. However, we personally find the labels `True` and `False` more readable.*

Section 4.2. Create index_filename and index_file_path

```
 1 # Now let's download and store a template file in our template directory.
 2 
 3 # This template file is written using the Jinja2 templating language, and will hav
 4 # dynamically updated at runtime serve for all web requests. (Traditionally, the d
 5 # returned by a web server is called 'index.html', hence the names below.)
 6 
 7 # Set the URL the file will be downloaded from.
 8 index_file_url = 'https://gist.githubusercontent.com/ilankham/d7608f36018daecc6baa
 9 
10 # Get the name of the file, and set the path where the file will be stored.
11 index_filename = Path(index_file_url).name
12 index_file_path = Path(template_dir, index_filename)
13 
14 # Download the file, and write it to the template directory.
15 index_file_response = requests.get(index_file_url)
16 index_file_path.write_text(index_file_response.text, encoding='utf-8')
17 
18 # Finally, print out some information about the file we downloaded.
19 print(f'The absolute path of the file we downloaded: {index_file_path.resolve()}')
20 print(f'The size of the file we downloaded: {index_file_path.stat().st_size:,} byt
```

```
The absolute path of the file we downloaded: /content/index.html
The size of the file we downloaded: 2,660 bytes
```

Concept Check 4.2

1. Short Answer: Given a URL, list some ways to download a file in SAS.
2. Fun Fact: `Path(template_dir, index_filename)` is equivalent to `template_dir / index_filename`

1. *Two common options are PROC HTTP and a `FILENAME` statement using the `URL` access method.*
2. *This Stack Overflow discussion has some good details: https://stackoverflow.com/a/68295436*

Section 4.3. List available librefs and the number of data tables in each

```
# TANGENT/ASIDE: We've now laid the groundwork for building and running our web ap
# we do, let's gather some information about our SAS session, which will be helpfu
# running our app.

# Iterate over all of the librefs available to the SAS Session created above, and
# of data tables available in each.
for libref in sorted(sas.assigned_librefs()):
    available_tables = [
        table for table in sas.list_tables(libref) if table[1] == 'DATA'
    ]
    print(f'libref: {libref}')
    print(f'number of tables: {len(available_tables)}')
    print('\n')
```

```
libref: MAPS
```

Concept Check 4.3

1. Short Answer: List some ways to get available librefs and their data tables in SAS.
2. Fun Fact: Any list comprehension (see Lines 8-10 for an example) can be turned into an equivalent for-loop. Also, just like for-loops, list comprehensions can be nested inside each other.

```
number of tables: 199
```

1. *Three common options are PROC CONTENTS, using special `sashelp` datasets in a DATA step, and using special `dictionary` datasets in a PROC SQL step.*
2. *This Stack Overflow discussion has some good details: https://stackoverflow.com/q/32429184*

```
libref: WORK
```

Section 4.4. Create parse_dataset_explorer_web_request

```
# We're now ready to define the "business logic" function for our web app, which w
# specific set of information, including contents from an HTTP Request, and return
# app will use when composing an HTTP Response.

# This function is intended to be free of "side effects", meaning it doesn't know
# web app or SAS session, other than what's passed to it via its arguments.
def parse_dataset_explorer_web_request(
    sas_session,
    sas_commands,
    output_dir,
    command,
    libref,
    libref_contents,
    dataset,
):

    if isinstance(libref_contents, str):
        libref_contents = json.loads(libref_contents.replace("'",'"'))

    return_value = {
        'libref': libref,
        'libref_contents': libref_contents,
        'dataset': dataset,
        'iframe_filename': '',
        'error': '',
    }

    if not command or command == 'reset page':
        pass
    elif command == 'submit libref':
        valid_librefs = map(str.lower, sas_session.assigned_librefs())
        libref_contents = sas_session.list_tables(libref)

        if not libref or libref.lower() not in valid_librefs:
            return_value['error'] = 'libref is invalid.'
        elif not libref_contents:
            return_value['error'] = 'libref is empty.'
        else:
            return_value['libref'] = libref
            return_value['libref_contents'] = [
                table[0] for table in libref_contents if table[1] == 'DATA'
            ]
    else:
        if not libref or not dataset:
            return_value['error'] = 'No dataset selected.'
        else:
            full_dataset_name = f'{libref}.{dataset}'
            iframe_filename = 'results-{}.html'.format(datetime.now().strftime('%Y
            iframe_file_path = output_dir / iframe_filename

            sas_output = sas_session.submit(sas_commands[command].format(dsn=full_
            iframe_file_path.write_text(sas_output['LST'], encoding='utf-8')

            return_value['iframe_filename'] = iframe_filename

    return return_value
```

Concept Check 4.4

1. True or False: `f'{libref}.{dataset}'` is equivalent to `'{}.{}'.format(libref,dataset)`
2. Short Answer: List some ways to create a user-defined function in SAS.

1. *True. There are (at least) four different string interpolation options in Python, with f-strings being the most recently added (as of Python 3.6). Since they were introduced, the Python community has largely coalesced on it being the best option. However, there is one case where f-strings don't work, and folks typically use `str.format` instead: When we want to put placeholders in a string and fill in values later.*
2. *This Stack Overflow discussion has some good details: https://stackoverflow.com/q/32429184*

Section 4.5. Create dataset_explorer_flask_app

```python
# And finally, we're ready to stand up our web app!

# Define a flask web app, and set it up to run through ngrok (a reverse proxy serv
# it'll be available over the public Internet.
dataset_explorer_flask_app = Flask(
    __name__,
    static_folder=sas_output_dir,
    template_folder=template_dir,
)
run_with_ngrok(dataset_explorer_flask_app)

# Define the SAS commands we want our web app to submit to a SAS session on our be
# string-templating placeholders standing in for what will be dynamically populate
# user-selected datasets.
SAS_COMMANDS = {
    'run proc contents': 'proc contents order=varnum data={dsn}; run;',
    'run proc freq': 'proc freq nlevels data={dsn}; tables _CHAR_; run;',
    'run proc means': 'proc means maxdec=2 data={dsn};  run;',
}

# Register a handler for an HTTP route for our web app, meaning the function below
# whenever someone tries to make a GET request (the default HTTP access method) at
# of our website.
@dataset_explorer_flask_app.route('/', methods=['GET'])
def handle_root_get_request():
    command = request.args.get('command','').lower()
    libref = request.args.get('libref')
    libref_contents = request.args.get('libref_contents')
    dataset = request.args.get('dataset')

    response_contents = parse_dataset_explorer_web_request(
        sas,
        SAS_COMMANDS,
        sas_output_dir,
        command,
        libref,
        libref_contents,
        dataset
    )

    return render_template(index_filename, **response_contents)

# Run the web app, and look for a ngrok.io URL in the resulting output; visiting t
# anyone with an Internet connection to interact with the app.
dataset_explorer_flask_app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://e091-35-237-88-141.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

Concept Check 4.5

1. Short Answer: What does the Python object `__main__` remind you of in SAS?
2. Short Answer: What type of Python object is `SAS_COMMANDS`?

1. *In Python, special variables that get created automatically tend to have their names surrounded by double underscores. Similarly, in SAS, special variables that get created automatically (e.g, `_N_` in a DATA step) tend to have their names surrounded by single underscores.*
2. *`SAS_COMMANDS` is a dictionary, which maps strings to strings.*

Section 4.6. Additional Exercises

For practice, we recommend the following:

- Run the code cell below to see an example of a simple, self-contained Flask web app that doesn't depend on any external functions for its "business logic."
- Modify the string on Lines 16-18 to change the (hard-coded) SAS output served by the web app.

```
 1 # Define a flask web app, and set it up to run through ngrok.
 2 simple_self_contained_flask_app = Flask(
 3     __name__,
 4     static_folder=sas_output_dir,
 5     template_folder=template_dir,
 6 )
 7 run_with_ngrok(simple_self_contained_flask_app)
 8 
 9 # Register a handler for an HTTP route for our web app.
10 @simple_self_contained_flask_app.route('/', methods=['GET'])
11 def handle_root_get_request():
12 
13     sas_submit_results = sas.submit(
14         '''
15             ods text='Hello, SAS ODA!';
16         '''
17     )
18 
19     return sas_submit_results['LST']
```

```

# Run the app, and look for a ngrok.io URL in the resulting output.
simple_self_contained_flask_app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://0787-35-237-88-141.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

```
# Define a flask web app, and set it up to run through ngrok.
simple_self_contained_flask_app = Flask(
    __name__,
    static_folder=sas_output_dir,
    template_folder=template_dir,
)
run_with_ngrok(simple_self_contained_flask_app)

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

    sas_submit_results = sas.submit(
        '''
            proc print data=sashelp.class( obs=7 keep=Name Age Height );
            run;
        '''
    )

    return sas_submit_results['LST']

# Run the app, and look for a ngrok.io URL in the resulting output.
simple_self_contained_flask_app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://7ae8-35-237-88-141.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

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
2. Try using Python outside of Google Colab. For example, if you're interested in setting up a local SASPy enviroment in order to have Python talk to a commercial SAS installation, you're welcome to follow setup instructions (see page 2) from a previous iteration of this course.
3. 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
4. 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 built-in functions like `isinstance` and `map`, see https://docs.python.org/3/library/functions.html
2. For more about the `datatime` package, see https://docs.python.org/3/library/datetime.html
3. For more about the `flask` package, see https://flask.palletsprojects.com/
4. For more about the `flask-ngrok` package, see https://github.com/gstaff/flask-ngrok

5. For more about the `json` package, see https://docs.python.org/3/library/json.html
6. For more about the `pathlib` package, see https://docs.python.org/3/library/pathlib.html
7. For more about the `requests` package, see https://docs.python-requests.org/
8. For more about the `saspy` package, including the methods used above, see the following:
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.assigned_librefs
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.list_tables
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.submit
9. For more about some of the Python features used, such as functions, list comphrensions, and control flow with if-then-else conditionals and for-loops, we recomend the following chapters of A Whirlwind Tour of Python:
    - https://jakevdp.github.io/WhirlwindTourOfPython/07-control-flow-statements.html
    - https://jakevdp.github.io/WhirlwindTourOfPython/08-defining-functions.html
    - https://jakevdp.github.io/WhirlwindTourOfPython/11-list-comprehensions.html
10. For more information on f-strings (i.e., Python strings like `f'https://httpstatuses.com/{defns_of_set_response.status_code}'`), see https://realpython.com/python-f-strings/.
11. 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