Everything is Better with Friends

Using SAS in Python Applications with SASPy and Open-Source Tooling (Getting Started)

Section 0. Setup and Connect to SAS OnDemand for Academics (ODA)

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 a value like `Asia Pacific 1`, `Asia Pacific 2`, `Europe 1`, `United States 1`, or `United States 2` next to your username in the upper-right corner. (For more information about Regions and using Python in Jupyter Notebooks, please see the ODA documentation at https://support.sas.com/ondemand/caq_new.html#region and https://support.sas.com/ondemand/saspy.html.)
2. If your ODA account is associated with a Region other than `United States 1`, comment out Line 11 by adding a number sign (`#`) at the beginning of the line, and then uncomment the list of servers corresponding to your Region.

    Note: As of the time of creation of this Notebook, only the Regions listed below were available. If your

SAS ODA account is associated with a Region that's not listed, you will need to manually add the appropriate servers.

3. Click anywhere in the code cell, and run the cell using Shift-Enter.
4. At the prompt `Please enter the OMR user id`, enter either your SAS ODA user ID or the email address associated with your ODA account. (Warning: Credentials are case sensitive!)
5. At the prompt `Please enter the password for OMR user`, enter the password for your SAS ODA account.

```
!pip install saspy

import saspy

sas = saspy.SASsession(
    java='/usr/bin/java',
    iomport=8591,
    encoding='utf-8',

    # For Region "United States 1", uncomment the line below.
    iomhost = ['odaws01-usw2.oda.sas.com','odaws02-usw2.oda.sas.com','odaws03-usw2.oda.sas.com','o

    # For Region "United States 2", uncomment the line below.
    #iomhost = ['odaws01-usw2-2.oda.sas.com','odaws02-usw2-2.oda.sas.com'],

    # For Region "Europe 1", uncomment the line below.
    #iomhost = ['odaws01-euw1.oda.sas.com','odaws02-euw1.oda.sas.com'],

    # For Region "Asia Pacific 1", uncomment the line below.
    #iomhost = ['odaws01-apse1.oda.sas.com','odaws02-apse1.oda.sas.com'],

    # For Region "Asia Pacific 2", uncomment the line below.
    #iomhost = ['odaws01-apse1-2.oda.sas.com','odaws02-apse1-2.oda.sas.com'],

)
print(sas)
```

```
Collecting saspy
  Downloading saspy-5.4.3-py3-none-any.whl (9.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 21.5 MB/s eta 0:00:00
Installing collected packages: saspy
Successfully installed saspy-5.4.3
Using SAS Config named: default
Please enter the OMR user id: matthew.t.slaughter@kpchr.org
Please enter the password for OMR user : ··········
SAS Connection established. Subprocess id is 594

Access Method         = IOM
SAS Config name       = default
SAS Config file       = /usr/local/lib/python3.10/dist-packages/saspy/sascfg.py
WORK Path             = /saswork/SAS_workB007000143AC_odaws02-usw2.oda.sas.com/SAS_work3248000143
SAS Version           = 9.04.01M7P08062020
SASPy Version         = 5.4.3
Teach me SAS          = False
```

```
Batch                 = False
Results               = Pandas
SAS Session Encoding  = utf-8
Python Encoding value = utf-8
SAS process Pid value = 82860
```

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 rich module for colorful printing
!pip install rich

# We'll use IPython to display DataFrames or HTML content
from IPython.display import display, HTML

# We'll use the pandas package to create and manipulate DataFrame objects
import pandas

# We'll use the platform platform to get information about our Python environment.
import platform

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

# We're also setting the maximum line width of rich.print to be a bit wider (to avoid line wrappin
from rich import get_console
console = get_console()
console.width = 165
```

```
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (13.6.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdo
```

Section 1. Setup and Connect to SAS OnDemand for Academics (ODA)

Example 1.1. Meet the Python environment

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
!cat /etc/*release*
print('\n')
print('Python Version:', platform.sys.version)
print('\n')
print(sorted(list(platform.sys.modules)))
```

```
!cat /etc/*release*
print('\n')
print('Python Version:', platform.sys.version)
print('\n')
print(sorted(list(platform.sys.modules)))
```

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.2 LTS"
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy

Python Version: 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]

[
    'IPython',
    'IPython.core',
    'IPython.core.alias',
    'IPython.core.application',
    'IPython.core.async_helpers',
    'IPython.core.autocall',
    'IPython.core.builtin_trap',
    'IPython.core.compilerop',
    'IPython.core.completer',
    'IPython.core.completerlib',
    'IPython.core.crashhandler',
    'IPython.core.debugger',
    'IPython.core.display',
    'IPython.core.display_trap',
    'IPython.core.displayhook',
    'IPython.core.displaypub',
    'IPython.core.error',
    'IPython.core.events',
    'IPython.core.excolors',
    'IPython.core.extensions',
    'IPython.core.formatters',
    'IPython.core.getipython',
    'IPython.core.history',
    'IPython.core.hooks',
    'IPython.core.inputsplitter',
    'IPython.core.inputtransformer',
    'IPython.core.inputtransformer2',
    'IPython.core.interactiveshell',
    'IPython.core.latex_symbols',
    'IPython.core.logger',
    'IPython.core.macro',
    'IPython.core.magic',
    'IPython.core.magic_arguments',
    'IPython.core.magics',
    'IPython.core.magics.auto',
```

Notes about Example 1.1.

1. Assuming a Python 3 kernel is associated with this Notebook, the following should be printed, separated by blank lines:
    - operating-system information
    - the Python version
    - a sorted list of python modules currently installed

2. This example illustrates three ways Python syntax differs from SAS:
    - We don't need semicolons at the end of each statement.
    - The code `PRINT('\n')` would produce an error because capitalization matters.
    - The code `platform.sys.version` uses object-oriented dot-notation to have the `platform` object module invoke the sub-module object `sys` nested inside of it, and then have `sys` invoke the object `version` nested inside of it. (Think Russian nesting dolls or turduckens.)
3. Python comes with a large standard library because of its "batteries included" philosophy, and numerous third-party modules are also actively developed and made freely available through sites like https://github.com/ and https://pypi.org/. For the examples in this notebook, we'll need these third-party modules:
    - `IPython`, which stands for "Interactive Python." Google Colab is built on top of JupyterLab, and JupyterLab is built on top of `IPython`, so `IPython` is already available in Google Colab.
    - `pandas`, which provided `DataFrame` objects. DataFrames can be found in other languages, like R, and are similar to SAS datasets. Because `pandas` is a fundamental package for working with data in Python, it's already available in Google Colab.
    - `saspy`, which is a Python package developed by the SAS Institute for connecting to a SAS kernel. Because `saspy` doesn't come pre-installed in Google Colab sessions, we had to manually install it in Section 0 above.
4. To increase performance, only a small number of modules in Python's standard library are available by default, which is why we needed to explicitly load the modules we'll be using in Section 0 above.
5. For extra credit, try the following:
    - Run the Python code `help(saspy)` to get the built-in help for the package `saspy` we'll be using in Sections 2-4 below.

Example 1.2. Python lists and indexing

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
hello_world_list = ['Hello', 'list']
print(hello_world_list)
print('\n')
print(type(hello_world_list))
```

```
1 hello_world_list = ['Hello', 'list']
2 print(hello_world_list)
3 print('\n')
4 print(type(hello_world_list))
```

```
['Hello', 'list']


<class 'list'>
```

```
<class 'list'>
```

Notes about Example 1.2.

1. A list object named `hello_world_list` with two string values is created, and the following are printed with a blank line between them:
    - the value of the list
    - its type (which is `<class 'list'>`)
2. Lists are a fundamental Python data structure and are similar to SAS DATA step arrays. Values in lists are always kept in insertion order, meaning the order they appear in the list's definition, and they can be individually accessed using numerical indexes within bracket notation:
    - `hello_world_list[0]` returns `'Hello'`
    - `hello_world_list[1]` returns `'list'`
3. This example illustrates another way Python syntax differs from SAS: The left-most element of a list is always at index `0`. Unlike SAS, customized indexing is only available for more sophisticated Python data structures, like the dictionaries and DataFrames we'll be using in the following examples.
4. For extra credit, try any or all of the following:
    - Print out the initial element of the list.
    - Print out the final element of the list.
    - Create a list of length five, and print its middle elements.

Example 1.3 Python dictionaries

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
hello_world_dict = {
        'salutation'     : ['Hello'       , 'dict'],
        'valediction'    : ['Goodbye'     , 'list'],
        'part of speech' : ['interjection', 'noun'],
}
print(hello_world_dict)
print('\n')
print(type(hello_world_dict))
```

```
1 hello_world_dict = {
2         'salutation'     : ['Hello'       , 'dict'],
3         'valediction'    : ['Goodbye'     , 'list'],
4         'part of speech' : ['interjection', 'noun'],
5 }
6 print(hello_world_dict)
7 print('\n')
```

```
print(type(hello_world_dict))
```

```
{'salutation': ['Hello', 'dict'], 'valediction': ['Goodbye', 'list'], 'part of speech': ['interje
<class 'dict'>
```

Notes about Example 1.3.

1. A dictionary (`dict` for short) object named `hello_world_dict` with three key-value pairs is created, and the following are printed with a blank line between them:
    - the value of the dictionary
    - its type (which is `<class 'dict'>`)
2. Dictionaries are another fundamental Python data structure and are related to SAS formats and DATA step hash tables. Dictionaries are more generally called *associative arrays* or *maps* because they map keys (appearing before the colons) to values (appearing after the colons). In other words, the value associated with each key can be accessed using bracket notation:
    - `hello_world_dict['salutation']` returns `['Hello', 'dict']`
    - `hello_world_dict['valediction']` returns `['Goodbye', 'list']`
    - `hello_world_dict['part of speech']` returns `['interjection', 'noun']`
3. Whenever indexable data structures are nested in Python, indexing methods can be combined. Here's an example combining dictionary indexing with list indexing: `hello_world_dict['salutation'][0] == ['Hello', 'dict'][0] == 'Hello'`.
4. For extra credit, try any or all of the following:
    - Print out the list with key `'salutation'`.
    - Print out the initial element in the list associated with key `'valediction'`.
    - Print out the final element in the list associated with key `'part of speech'`.

Example 1.4 Pandas DataFrames

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
hello_world_df = pandas.DataFrame(
    {
        'salutation'     : ['Hello'      , 'DataFrame'],
        'valediction'    : ['Goodbye'    , 'dict'],
        'part of speech' : ['exclamation', 'noun'],
    }
)
display(hello_world_df)
print('\n')
print(hello_world_df.shape)
print('\n')
```

```
hello_world_df.info()
```

```
hello_world_df = pandas.DataFrame(
    {
        'salutation'     : ['Hello'      , 'DataFrame'],
        'valediction'    : ['Goodbye'    , 'dict'],
        'part of speech' : ['exclamation', 'noun'],
    }
)
display(hello_world_df)
print('\n')
print(hello_world_df.shape)
print('\n')
hello_world_df.info()
```

	salutation	valediction	part of speech
0	Hello	Goodbye	exclamation
1	DataFrame	dict	noun

```
(2, 3)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   salutation      2 non-null      object
 1   valediction     2 non-null      object
 2   part of speech  2 non-null      object
dtypes: object(3)
memory usage: 176.0+ bytes
```

Notes about Example 1.4.

1. A DataFrame (`df` for short) object named `hello_world_df` with 2 rows and 3 columns is created, and the following are printed with blank lines between them:
    - the values in the DataFrame
    - the number of rows and columns in `hello_world_df`
    - some information about the DataFrame, which is obtained by having `hello_world_df` calling its `info` method
2. Since DataFrames aren't built into Python, we had to import the `pandas` module at the start of this notebook. Like their R counterpart, DataFrames are two-dimensional arrays of values comparable to SAS datasets. However, while SAS datasets are typically accessed from disk and processed row-by-row, DataFrames are loaded into memory all at once. This means values in DataFrames can be randomly accessed, but it also means the size of DataFrames can't grow beyond available memory.
3. The dimensions of the DataFrame are determined as follows:
    - The keys `'salutation'`, `'valediction'`, and `'part of speech'` of the dictionary passed to the

`DataFrame` constructor function become column labels.
    - Because each key maps to a list of length two, each column will be two elements tall. (Note: An error will occur if the lists are not the same length).
4. This example gives one option for building a DataFrame, but the constructor function accepts many object types, including nested lists or another DataFrame. See https://pandas.pydata.org/docs/
5. For extra credit, try any or all of the following:
    - Print out the column with key `'salutation'`.
    - Print out the initial element in the column with key `'valediction'`.
    - Print out the final element in the column with key `'part of speech'`.

      Hint: DataFrame columns can be indexed just like dictionaries, and their rows can be indexed numerically like lists.

Section 2. SASPy Data Round Trip

Example 2.1. Load a SAS dataset into a pandas DataFrame

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
fish_df_smelt_only = sas.sasdata2dataframe(
    table='fish',
    libref='sashelp',
    dsopts={
        'where' : ' Species = "Smelt" ',
        'obs'   : 10,
    },
)
print(type(fish_df_smelt_only))
print('\n')
print(fish_df_smelt_only.shape)
print('\n')
display(fish_df_smelt_only.head())
```

```
fish_df_smelt_only = sas.sasdata2dataframe(
    table='fish',
    libref='sashelp',
    dsopts={
        'where' : ' Species = "Smelt" ',
        'obs'   : 10,
    },
)
print(type(fish_df_smelt_only))
```

```
print('\n')
print(fish_df_smelt_only.shape)
print('\n')
display(fish_df_smelt_only.head())
```

```
<class 'pandas.core.frame.DataFrame'>

(10, 7)
```

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Smelt	6.7	9.3	9.8	10.8	1.7388	1.0476
1	Smelt	7.5	10.0	10.5	11.6	1.9720	1.1600
2	Smelt	7.0	10.1	10.6	11.6	1.7284	1.1484
3	Smelt	9.7	10.4	11.0	12.0	2.1960	1.3800
4	Smelt	9.8	10.7	11.2	12.4	2.0832	1.2772

Notes about Example 2.1.

1. A DataFrame object named `fish_df_smelt_only` is created from the first 10 rows of the SAS dataset `fish` in the `sashelp` library satisfying `Species = "Smelt"`, and the following are printed with a blank line between them:
    - the type of object `fish_df_smelt_only` (which is `<class 'pandas.core.frame.DataFrame'>`)
    - the number of rows and columns in `fish_df_smelt_only`
    - the first five rows of `fish_df_smelt_only`, which are at row indices 0 through 4 since Python uses zero-based indexing
2. The `sas` object represents a connection to a SAS session and was created in Section 0 above. Here, `sas` uses its `sasdata2dataframe` method to access the SAS library `sashelp` and load the contents of `sashelp.fish(obs=10 where=(Species = "Smelt"))` into `fish_df_smelt_only`.
3. For extra credit, try any or all of the following:
    - Pass a numerical parameter to the `head` method to see a different number of rows (e.g., `fish_df_smelt_only.head(7)`).
    - Change the `head` method to `tail` to see a different part of the dataset.
    - To view other portions of `fish_df_smelt_only`, explore the more advanced indexing methods `loc` and `iloc` explained at https://brohrer.github.io/dataframe_indexing.html

Example 2.2. Manipulate a DataFrame

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
fish_df     = sas.sasdata2dataframe(table='fish',libref='sashelp')
fish_df_g   = fish_df.groupby('Species')
fish_df_gs  = fish_df_g['Weight']
```

```
fish_df_gsa = fish_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
display(fish_df_gsa)
```

```
1 fish_df     = sas.sasdata2dataframe(table='fish',libref='sashelp')
2 fish_df_g   = fish_df.groupby('Species')
3 fish_df_gs  = fish_df_g['Weight']
4 fish_df_gsa = fish_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
5 display(fish_df_gsa)
```

Species	count	std	mean	min	max
Bream	34	206.604585	626.000000	242.0	1000.0
Parkki	11	78.755086	154.818182	55.0	300.0
Perch	56	347.617717	382.239286	5.9	1100.0
Pike	17	494.140765	718.705882	200.0	1650.0
Roach	20	88.828916	152.050000	0.0	390.0
Smelt	14	4.131526	11.178571	6.7	19.9
Whitefish	6	309.602972	531.000000	270.0	1000.0

Notes about Example 2.2.

1. The DataFrame `fish_df` is created from the SAS dataset `sashelp.fish`, and this time all 159 rows are included since no dataset options were used. After some `pandas` operations are performed on `fish_df`, the following is printed:
    - a table giving the number of rows, standard deviation, mean, min, and max of `Weight` in `fish_df` when aggregated by `Species`
2. This is accomplished by creating a series of new DataFrames:
    - The DataFrame `fish_df_g` is created from `fish_df` using the `groupby` method to group rows by values in column `'Species'`.
    - The DataFrame `fish_df_gs` is created from `fish_df_g` by extracting the `'Weight'` column using bracket notation.
    - The DataFrame `fish_df_gsa` is created from `fish_df_gs` using the `agg` method to aggregate by the functions in the list `['count', 'std', 'mean', 'min', 'max']`.
3. Identical results could be obtained using the following SAS code:

```
PROC MEANS DATA=sashelp.fish STD MEAN MIN MAX;
    CLASS species;
    VAR weight;
RUN;
```

However, while PROC MEANS operates on SAS datasets row-by-row from disk, DataFrames are stored entirely in main memory. This allows any number of DataFrame operations to be combined for on-the-fly reshaping using "method chaining." In other words, `fish_df_gsa` could instead be created with the following one-liner, which avoids the need for intermediate DataFrames (and executes much more quickly):

```
fish_df.groupby('Species')['Weight'].agg(['count', 'std', 'mean', 'min', 'max'])
```

4. For extra credit, try any or all of the following:
    - Move around and/or remove functions used for aggregation, and see how the output changes.
    - Change the variable whose values are summarized to `'Width'`.
    - Obtain execution time for the one-liner version by included the JupyerLab magic `%%time` at the start of a code cell (on a line by itself).

Example 2.3. Load a DataFrame into a SAS dataset

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
sas.dataframe2sasdata(
    fish_df_gsa.reset_index(),
    table="fish_sds_gsa",
    libref="work"
)
sas_submit_return_value = sas.submit(
    '''
        PROC PRINT DATA=fish_sds_gsa;
        RUN;
    '''
)
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
sas_submit_results = sas_submit_return_value['LST']
display(HTML(sas_submit_results))
```

```
1 sas.dataframe2sasdata(
2     fish_df_gsa.reset_index(),
3     table="fish_sds_gsa",
4     libref="work"
5 )
6 sas_submit_return_value = sas.submit(
7     '''
8         PROC PRINT DATA=fish_sds_gsa;
```

```
        RUN;
    '''
)
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
sas_submit_results = sas_submit_return_value['LST']
display(HTML(sas_submit_results))
```

```
33                                                      The SAS System                       S

261        ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inl
261      ! ods graphics on / outputfmt=png;
262
263
264                PROC PRINT DATA=fish_sds_gsa;
265                RUN;
266
267
268
269        ods html5 (id=saspy_internal) close;ods listing;
270

34                                                      The SAS System                       S

271
```

The SAS System

Obs	Species	count	std	mean	min	max
1	Bream	34	206.605	626.000	242.0	1000.0
2	Parkki	11	78.755	154.818	55.0	300.0
3	Perch	56	347.618	382.239	5.9	1100.0
4	Pike	17	494.141	718.706	200.0	1650.0
5	Roach	20	88.829	152.050	0.0	390.0
6	Smelt	14	4.132	11.179	6.7	19.9
7	Whitefish	6	309.603	531.000	270.0	1000.0

Notes about Example 2.3.

1. The DataFrame `fish_df_gsa`, which was created in Example 2.2 from the SAS dataset `sashelp.fish`, is used to create the new SAS dataset `work.fish_sds_gsa`. The SAS PRINT procedure is then called, and the following is displayed:
    - the SAS log returned by PROC PRINT
    - the output returned by PROC PRINT
2. The `sas` object, which was created in Section 0, is a persistent connection to a SAS session, and two of its methods are used as follows:
    - The `dataframe2sasdata` method writes the contents of the DataFrame `fish_df_gsa` to the SAS dataset `fish_sds_gsa` stored in the `Work` library. (Note: The row indexes of the DataFrame `fish_df_gsa` are lost when the SAS dataset `fish_sds_gsa` is created.)
    - The `submit` method is used to submit the PROC PRINT step to the SAS kernel, and a dictionary is

returned with the following two key-value pairs:

   - `sas_submit_return_value['LOG']` is a string comprising the plain-text log resulting from executing PROC PRINT
   - `sas_submit_return_value['LST']` is a string comprising the results from executing PROC PRINT, which will be in HTML by default.

3. Python strings surrounded by single quotes (e.g., `'Hello, World!'`) cannot be written across multiple lines of code, whereas strings surrounded by triple quotes (e.g., the argument to the `submit` method) can.

4. For extra credit, try any or all of the following:

   - Change the SAS procedure used to interact with SAS dataset `work.fish_sds_gsa` (e.g., try PROC CONTENTS).
   - Change the format of the SAS output by adding the argument `results='TEXT'` to the `sas.submit` call, and display it with the `print` function instead.

Section 3. Executing SAS Procedures with Convenience Methods

Example 3.1. Connect directly to a SAS dataset

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
fish_sds = sas.sasdata(table='fish', libref='sashelp')
print(type(fish_sds))
print('\n')
display(fish_sds.columnInfo())
print('\n')
display(fish_sds.means())
```

```
1 fish_sds = sas.sasdata(table='fish', libref='sashelp')
2 print(type(fish_sds))
3 print('\n')
4 display(fish_sds.columnInfo())
5 print('\n')
6 display(fish_sds.means())
```

```
<class 'saspy.sasdata.SASdata'>
```

	Member	Num	Variable	Type	Len	Pos
0	SASHELP.FISH	6.0	Height	Num	8.0	32.0
1	SASHELP.FISH	3.0	Length1	Num	8.0	8.0
2	SASHELP.FISH	4.0	Length2	Num	8.0	16.0

3	SASHELP.FISH	5.0	Length3	Num	8.0	24.0
4	SASHELP.FISH	1.0	Species	Char	9.0	48.0
5	SASHELP.FISH	2.0	Weight	Num	8.0	0.0
6	SASHELP.FISH	7.0	Width	Num	8.0	40.0

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P50	P75	
0	Weight	158.0	1.0	272.5000	398.695570	359.086204	0.0000	120.0000	272.5000	650.0000	16
1	Length1	159.0	0.0	25.2000	26.247170	9.996441	7.5000	19.0000	25.2000	32.7000	
2	Length2	159.0	0.0	27.3000	28.415723	10.716328	8.4000	21.0000	27.3000	36.0000	
3	Length3	159.0	0.0	29.4000	31.227044	11.610246	8.8000	23.1000	29.4000	39.7000	
4	Height	159.0	0.0	7.7860	8.970994	4.286208	1.7284	5.9364	7.7860	12.3778	
5	Width	159.0	0.0	4.2485	4.417486	1.685804	1.0476	3.3756	4.2485	5.5890	

Notes about Example 3.1.

1. The SASdata object `fish_sds` (meaning a direct connection to the disk-based SAS dataset `sashelp.fish` in our remote SAS ODA session, not an in-memory DataFrame in our local Python session) is created, and the following are printed with a blank line between them:
    - the type of object `fish_sds` (which is `<class 'saspy.sasdata.SASdata'>`)
    - a portion of PROC CONTENTS applied to the SAS dataset `sashelp.fish`
    - summary information about the numeric columns in `sashelp.fish`
2. The `sas` object, which was created in Section 0, is a persistent connection to a SAS session, and its `sasdata` method is used to create the connection to `sashelp.fish`.
3. The `fish_sds` object calls its *convenience method* `means`, which implicitly invokes PROC MEANS on `sashelp.fish`.
4. For extra credit, try the following:
    - Explore the additional convenience methods listed at https://sassoftware.github.io/saspy/api.html#sas-data-object.

Example 3.2 Display generated SAS code

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
sas.teach_me_SAS(True)
fish_sds.means()
sas.teach_me_SAS(False)
```

```
sas.teach_me_SAS(True)
fish_sds.means()
sas.teach_me_SAS(False)
```

```
    proc means data=sashelp.'fish'n  stackodsoutput n nmiss median mean std min p25 p50 p75 max;run;
```

Notes about Example 3.2

1. The SASdata object `fish_sds`, which was created in Example 3.1 as a direct connection to the SAS dataset `sashelp.fish`, calls its *convenience method* `means` within a "Teach Me SAS" sandwich, and the following is printed:
    - the SAS code for the PROC MEANS step implicitly generated by the `means` convenience method
2. The `sas` object, which was created in Section 0, is a persistent connection to a SAS session, and its `teach_me_SAS` method is used as follows:
    - When called with argument `True`, SAS output is suppressed for all subsequent `saspy` convenience methods, and the SAS code that would be generated by the convenience method is printed instead.
    - When `teach_me_SAS` is called with argument `False`, this behavior is turned off.
3. `True` and `False` are standard Python objects. Like their SAS equivalents, they are interchangeable with the values `1` and `0`, respectively.
4. One benefit of this process is being able to extract and modify the SAS code. For example, if a convenience method doesn't offer an option like a class statement for PROC MEANS, we can manually add it to the code generated by the `teach_me_SAS` method and then execute the modified SAS code using the `submit` method (as in Example 2.3 above).
5. For extra credit, try any or all of the following:
    - Change `means` to a different convenience method, such as `columnInfo`.
    - Submit the generated SAS code by `teach_me_SAS` using the `submit` method.

Example 3.3 Adding variables to a SAS dataset

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
class_sds = sas.sasdata(
    table='class',
    libref='sashelp'
)
class_bmi_sds = sas.sasdata(
    table='class_bmi',
    libref='work'
)
class_sds.add_vars(vars = {'bmi':'(weight/height**2)*703'}, out = class_bmi_sds)
display(class_bmi_sds.head())
print('\n')
```

```
display(class_bmi_sds.means())
```

```
class_sds = sas.sasdata(
    table='class',
    libref='sashelp'
)
class_bmi_sds = sas.sasdata(
    table='class_bmi',
    libref='work'
)
class_sds.add_vars(vars = {'bmi':'(weight/height**2)*703'}, out = class_bmi_sds)
display(class_bmi_sds.head())
print('\n')
display(class_bmi_sds.means())
```

```
Table work.class_bmi does not exist. This SASdata object will not be useful until the data set is

77                                                      The SAS System                       S

532
533         data work.'class_bmi'n ; set sashelp.'class'n ;
534         bmi = (weight/height**2)*703;
535         ; run;
536
537
538

78                                                      The SAS System                       S

539
```

	Name	Sex	Age	Height	Weight	bmi
0	Alfred	M	14.0	69.0	112.5	16.611531
1	Alice	F	13.0	56.5	84.0	18.498551
2	Barbara	F	13.0	65.3	98.0	16.156788
3	Carol	F	14.0	62.8	102.5	18.270898
4	Henry	M	14.0	63.5	102.5	17.870296

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P50	P
0	Age	19.0	0.0	13.000000	13.315789	1.492672	11.000000	12.000000	13.000000	15.0000
1	Height	19.0	0.0	62.800000	62.336842	5.127075	51.300000	57.500000	62.800000	66.5000
2	Weight	19.0	0.0	99.500000	100.026316	22.773933	50.500000	84.000000	99.500000	112.5000
3	bmi	19.0	0.0	17.804511	17.863252	2.092619	13.490001	16.611531	17.804511	20.0943

Notes about Example 3.3.

1. The SASdata object `class_sds` (meaning a direct connection to the disk-based SAS dataset `sashelp.class` in our remote SAS ODA session) is created, the results of adding a new column to

`sashelp.class` in our remote SAS ODA session) is created, the results of adding a new column to `class_sds` are then output into the new SASdata object `class_bmi_sds`, and the following are printed with a blank line between them:

   - the first few rows of `class_bmi_sds`
   - summary information about the numeric columns in `class_bmi_sds`

2. The `sas` object, which was created in Section 0, is a persistent connection to a SAS session, and two of its methods are used as follows:

   - The `sasdata` method is used twice, first to create a pointer to `sashelp.class` and then to create a pointer to the not-yet-created SAS dataset `work.class_bmi`.
   - The `add_vars` method is used to creata a new column in `sashelp.class`, but to output the results of creating this new column as `work.class_bmi`. (If no `out=` argument has been specified, SAS would have attempted to modify `sashelp.class` in place.)

3. Identical results could be obtained using the following SAS code:

```
DATA work.class_bmi;
    SET sashelp.class;
    bmi = (weight/height**2)*703;
RUN;
PROC PRINT DATA=work.class_bmi(obs=5);
RUN;
PROC MEANS DATA=work.class_bmi;
RUN;
```

4. For extra credit, try any or all of the following:

   - Print only the first three rows of `class_bmi_sds` by adding a numeric argument to the `head` method call (just like the corresponding `pandas` method).
   - Add a `dsopts=` parameter to either use of the `sasdata` method above. (For example, you could use similar syntax as in Example 2.1 to limit the columns with `where` and/or `obs` options.)

     Note: Just like the `sasdata2dataframe` method used in Example 2.1, the `sasdata` method has a `dsopts` argument, which allows dataset options to specified. The underlying SAS dataset itself will not be modified unless `dsopts` is specified for an output dataset.

Section 4. Staying D.R.Y. (aka "Don't Repeat Yourself!")

Example 4.1. Imitate the SAS Macro Processor

Type the following into the code cell immediately below, and then run that cell using Shift-Enter:

```
sas_code_fragment = 'TITLE "Dataset: sashelp.{dsn}"; PROC MEANS DATA=sashelp.{dsn}; RUN; TITLE;'
for dataset_name in ['fish', 'class', 'iris']:
    sas_submit_return_value = sas.submit(
        sas_code_fragment.format(dsn=dataset_name)
    )
    print(sas_submit_return_value['LOG'])
    display(HTML(sas_submit_return_value['LST']))
```

```
1 sas_code_fragment = 'TITLE "Dataset: sashelp.{dsn}"; PROC MEANS DATA=sashelp.{dsn}; RUN; TITLE;'
2 for dataset_name in ['fish', 'class', 'iris']:
3     sas_submit_return_value = sas.submit(
4         sas_code_fragment.format(dsn=dataset_name)
5     )
6     print(sas_submit_return_value['LOG'])
7     display(HTML(sas_submit_return_value['LST']))
```

```
107                                                The SAS System                    S

722         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inl
722       ! ods graphics on / outputfmt=png;
723
724         TITLE "Dataset: sashelp.fish"; PROC MEANS DATA=sashelp.fish; RUN; TITLE;
725
726
727         ods html5 (id=saspy_internal) close;ods listing;
728

108                                                The SAS System                    S

729
```

Dataset: sashelp.fish

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Weight	158	398.6955696	359.0862037	0	1650.00
Length1	159	26.2471698	9.9964412	7.5000000	59.0000000
Length2	159	28.4157233	10.7163281	8.4000000	63.4000000
Length3	159	31.2270440	11.6102458	8.8000000	68.0000000
Height	159	8.9709937	4.2862076	1.7284000	18.9570000
Width	159	4.4174855	1.6858039	1.0476000	8.1420000

```
109                                                The SAS System                    S

732         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inl
732       ! ods graphics on / outputfmt=png;
733
734         TITLE "Dataset: sashelp.class"; PROC MEANS DATA=sashelp.class; RUN; TITLE;
735
736
737         ods html5 (id=saspy_internal) close;ods listing;
738

110                                                The SAS System                    S

739
```

Dataset: sashelp.class

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Age	19	13.3157895	1.4926722	11.0000000	16.0000000
Height	19	62.3368421	5.1270752	51.3000000	72.0000000
Weight	19	100.0263158	22.7739335	50.5000000	150.0000000

```
111                                                The SAS System                    S

742        ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inl
742      ! ods graphics on / outputfmt=png;
743
744        TITLE "Dataset: sashelp.iris"; PROC MEANS DATA=sashelp.iris; RUN; TITLE;
745
746
747        ods html5 (id=saspy_internal) close;ods listing;
748

112                                                The SAS System                    S

749
```

Dataset: sashelp.iris

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
SepalLength	Sepal Length (mm)	150	58.4333333	8.2806613	43.0000000	79.0000000
SepalWidth	Sepal Width (mm)	150	30.5733333	4.3586628	20.0000000	44.0000000
PetalLength	Petal Length (mm)	150	37.5800000	17.6529823	10.0000000	69.0000000
PetalWidth	Petal Width (mm)	150	11.9933333	7.6223767	1.0000000	25.0000000

Notes about Example 4.1.

1. A string object named `sas_code_fragment` is created with templating placeholder `{dsn}`, which will be filled using other strings in subsequent uses of `sas_code_fragment`.
2. The output of PROC MEANS applied to SAS datasets `sashelp.fish`, `sashelp.class`, and `sashelp.iris` is then displayed.
3. The `sas` object represents a connection to a SAS session and was created in Section 0. Here, `sas` calls its `submit` method for each value of the for-loop indexing variable `dsn`, and the `{data}` portion of `sas_code_fragment` is replaced by the value of `dsn`. In other words, the following SAS code is submitted to the SAS kernel:

```
TITLE "Dataset: sashelp.fish"; PROC MEANS DATA=sashelp.fish; RUN; TITLE;
TITLE "Dataset: sashelp.class"; PROC MEANS DATA=sashelp.class; RUN; TITLE;
TITLE "Dataset: sashelp.iris"; PROC MEANS DATA=sashelp.iris; RUN; TITLE;
```

4. The same outcome could also be achieved with the following SAS macro code:

```
%LET list_length = %sysfunc(countw(&dsn_list));
%DO i = 1 %TO &list_length;
    %LET dsn = %scan(&dsn_list.,&i.);
    TITLE "Dataset: sashelp.&dsn.";
    PROC MEANS DATA=sashelp.&dsn.;
    RUN;
    TITLE;
%END;
```

However, note the following differences:

- Python allows us to concisely repeat an arbitrary block of code by iterating over a list using a for-loop. In other words, the body of the for-loop (meaning everything indented underneath it, since Python uses indentation to determine scope) is repeated for each string in the list `['fish','class','iris']`.
- The SAS macro facility only provides do-loops based on numerical index variables (the macro variable `i` above), so clever tricks like implicitly defined arrays (macro variable `dsn_list` above) need to be used together with functions like `%scan` to extract a sequence of values.

5. For extra credit, try any or all of the following:
    - Add additional SASHELP datasets to the list being iterated over by the for-loop (e.g., iris or cars).
    - Change the `sas_code_fragment` to run a different SAS procedure (e.g., PROC PRINT).

Wrapping Up: Call to Action!

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
        - 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 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

1. 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
2. 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 `pandas` package, including the methods used above, see the following:
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html
    - https://pandas.pydata.org/docs/reference/api/pandas.Index.shape.html
2. For more about the `platform` package, see https://docs.python.org/3/library/platform.html
3. For more about the `rich` package, see https://rich.readthedocs.io/
4. For more about the `saspy` package, including the methods used above, see the following:
    - https://sassoftware.github.io/saspy/api.html#saspy.sasdata.SASdata.add_vars
    - https://sassoftware.github.io/saspy/api.html#saspy.sasdata.SASdata.columnInfo
    - https://sassoftware.github.io/saspy/api.html#saspy.sasdata.SASdata.head
    - https://sassoftware.github.io/saspy/api.html#saspy.sasdata.SASdata.means
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.dataframe2sasdata
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.sasdata2dataframe
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.submit
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.teach_me_SAS