Everything is Better with Friends

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

Setup for Part 1

A few notes before we get started...

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 by clicking on the link near the bottom of the ODA login page under the heading "Get Started".)
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. Some useful Zoom Reactions:
    - 👍 (Thumbs Up) when you're done with a section
    - ✋ (Raise Hand) when you need tech support
    - ☕ (I'm Away) to let us know you've stepped away
6. 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` at the top of screen near your username. (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 2`, comment out Line 40 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.
5. At the prompt `Please enter the password for OMR user`, enter the password for your SAS ODA account.

```
!pip install saspy

# import standard library packages
import io
import pathlib
import zipfile

# import third-party libraries
import requests
import saspy

# because of recent changes to SAS ODA, we may need to install some files in our Colab session
zip_file_url = 'https://drive.google.com/uc?id=1vQ6oVgky8UcLAvhct7CL8Oc5I9Mctiw5&export=download'
```

```
expected_zip_file_contents = {'sas.rutil.jar', 'sas.rutil.nls.jar', 'sastpj.rutil.jar'}
jar_file_installation_path = '/usr/local/lib/python3.8/dist-packages/saspy/java/iomclient/'

# check the JAVA config files currently available in the SASPy installation of our Colab session
current_saspy_jar_files = {
    file.name
    for file
    in pathlib.Path(jar_file_installation_path).glob('*.jar')
}

# if any of three specific .jar files aren't found, download and install them in our Colab session
if not expected_zip_file_contents.issubset(current_saspy_jar_files):
  zip_file_url_response = requests.get(zip_file_url)
  zip_file_contents = zipfile.ZipFile(io.BytesIO(zip_file_url_response.content))
  zip_file_contents.extractall('/usr/local/lib/python3.8/dist-packages/saspy/java/iomclient/')

# with the preliminaries out of the way, we can now establish a connection from Colab to SAS ODA
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','odaws04-usw2.

    # 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)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting saspy
  Downloading saspy-4.7.0.tar.gz (9.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 37.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: saspy
  Building wheel for saspy (setup.py) ... done
  Created wheel for saspy: filename=saspy-4.7.0-py3-none-any.whl size=9938036 sha256=70d6891ffb8cb864d3498ab5
  Stored in directory: /root/.cache/pip/wheels/c1/ab/9f/ffe82d792f6a8314e07be05a27f8fa0b8459e0110a682cf8c6
Successfully built saspy
Installing collected packages: saspy
Successfully installed saspy-4.7.0
Using SAS Config named: default
Please enter the OMR user id: isaiah.p.lankham@kpchr.org
Please enter the password for OMR user : ··········
SAS Connection established. Subprocess id is 473

Access Method         = IOM
SAS Config name       = default
SAS Config file       = /usr/local/lib/python3.8/dist-packages/saspy/sascfg.py
WORK Path             = /saswork/SAS_workF9D100014741_odaws02-usw2-2.oda.sas.com/SAS_work624800014741_odaws02
SAS Version           = 9.04.01M7P08062020
SASPy Version         = 4.7.0
Teach me SAS          = False
Batch                 = False
Results               = Pandas
SAS Session Encoding  = utf-8
Python Encoding value = utf-8
SAS process Pid value = 83777
```

Notes:

- This installs the SASPy package and establishes a connection from Python in Google Colab to a SAS session running in SAS ODA.

- If an error is displayed, an incompatible kernel has been chosen. This Notebook was developed using the Python 3.8 kernel provided in Google Colab as of February 2023.
- ODA was recently upgraded to SAS version 9.40M7, which is why SASPy needs the contents of a .zip file to be downloaded and installed inside Colab before SASPy can connect to SAS ODA. If Python and SAS were installed on the same machine, the contents of the .zip file would already be available as part of the SAS installation, per https://sassoftware.github.io/saspy/configuration.html
- If your SAS session times out or terminates (e.g., by closing this notebook or using the `sas.endsas()` command), you'll need to run this cell again and re-enter your ODA login credentials.

Install and import additional packages

```
# Install the rich module for colorful printing, limiting the version for compatibility with Colab
!pip install 'rich<13.3'

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

# 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 wrapping)
from rich import get_console
console = get_console()
console.width = 165
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting rich<13.3
  Downloading rich-13.2.0-py3-none-any.whl (238 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 238.9/238.9 KB 9.0 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions<5.0,>=4.0.0 in /usr/local/lib/python3.8/dist-packages (from
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in /usr/local/lib/python3.8/dist-packages (from rich<13
Collecting markdown-it-py<3.0.0,>=2.1.0
  Downloading markdown_it_py-2.2.0-py3-none-any.whl (84 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.5/84.5 KB 12.2 MB/s eta 0:00:00
Collecting mdurl~=0.1
  Downloading mdurl-0.1.2-py3-none-any.whl (10.0 kB)
Installing collected packages: mdurl, markdown-it-py, rich
Successfully installed markdown-it-py-2.2.0 mdurl-0.1.2 rich-13.2.0
```

Part 1. Calling SAS/STAT procedures in Python applications

Section 1.1. Create titanic_sds

```
# I'd rather not have spaces and slashes in my SAS variable names.
sas.submit('options validvarname=v7;')

# Read the titanic dataset into SAS from a URL.
titanic_url = 'https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv'
titanic_sds = sas.read_csv(file=titanic_url,table='titanic',libref='work')

# What kind of object is titanic_sds?
print(type(titanic_sds))
print('\n')
print(titanic_sds)

# Curious what's under the hood?
print('Here\'s the SAS code being used by SASPy to import the dataset:')
sas.teach_me_SAS(True)
sas.read_csv(file=titanic_url,table='titanic',libref='work')
sas.teach_me_SAS(False)
print('\n')

# Let's take a look at the data.
display(titanic_sds.columnInfo())
display(titanic_sds.head())
```

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

Libref  = work
Table   = titanic
Dsopts  = {}
Results = Pandas

Here's the SAS code being used by SASPy to import the dataset:
filename x url "https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv";
proc import datafile=x out=work.'titanic'n dbms=csv replace;  run;
```

	Member	Num	Variable	Type	Len	Pos	Format	Informat
0	WORK.TITANIC	5.0	Age	Num	8.0	16.0	BEST12.	BEST32.
1	WORK.TITANIC	8.0	Fare	Num	8.0	40.0	BEST12.	BEST32.
2	WORK.TITANIC	3.0	Name	Char	54.0	48.0	$54.	$54.
3	WORK.TITANIC	7.0	Parents_Children_Aboard	Num	8.0	32.0	BEST12.	BEST32.
4	WORK.TITANIC	2.0	Pclass	Num	8.0	8.0	BEST12.	BEST32.
5	WORK.TITANIC	4.0	Sex	Char	6.0	102.0	$6.	$6.
6	WORK.TITANIC	6.0	Siblings_Spouses_Aboard	Num	8.0	24.0	BEST12.	BEST32.
7	WORK.TITANIC	1.0	Survived	Num	8.0	0.0	BEST12.	BEST32.

	Survived	Pclass	Name	Sex	Age	Siblings_Spouses_Aboard	Parents_Chil
0	0.0	3.0	Mr. Owen Harris Braund	male	22.0	1.0	
1	1.0	1.0	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1.0	
2	1.0	3.0	Miss. Laina Heikkinen	female	26.0	0.0	
3	1.0	1.0	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1.0	
4	0.0	3.0	Mr. William Henry Allen	male	35.0	0.0	

Concept Check 1.1

- Try this, and see what happens: Comment out Line 17 so that `sas.teach_me_SAS` is no longer turned off, and then rerun the code cell above.
- True or False: Commenting out Line 17, so that `sas.teach_me_SAS` is left on, has no effect on subsequent code.
- Fun Fact: The SASdata object `titanic_sds` does not represent a dataset kept in memory in our local Google Colab session. Instead, it's a pointer to normal SAS dataset file kept on disk in the remote SAS ODA session.

Solution: False! If we don't turn "Teach Me SAS" off, Lines 21-22 won't execute.

Section 1.2. Create titanic_partitions

```
 1 # Make sure sas.teach_me_SAS is set to False.
 2 sas.teach_me_SAS(False)
 3
 4 # Let's partition the dataset into subsets for training and testing a predictive model.
 5 titanic_partitions = titanic_sds.partition(seed=42, singleOut=False)
 6
 7 # And then print some information about each subset.
 8 print(type(titanic_partitions))
 9 print('\n')
10 print(titanic_partitions)
11 print('\n')
12 print(f'{titanic_partitions[0].obs()} observations for training')
13 print(f'{titanic_partitions[1].obs()} observations for test')
```

```
<class 'tuple'>

(Libref  = work
Table   = titanic1_train
```

Concept Check 1.2

- Try this, and see what happens: Try using `sas.teach_me_SAS` to see the SAS code generated by the `partition` method on Line 5.
- True or False: In order to make sure `titanic_partitions` is created as expected (without a Traceback, a.k.a., a run-time error), Line 5 needs to be run with `sas.teach_me_SAS` turned off.
- Fun Fact: A `tuple` is like a fixed-length `list`. Whereas a `list` can be changed in place, and can grow and shrink in size, a `tuple` is immutable. This means any attempt to change a `tuple` actually results in a new `tuple` being created, instead. (Other examples of immutable Python objects include `int` and `str`.)

Solution: False! If we don't turn "Teach Me SAS" off, Lines 8-13 will result in Traceback errors since the variable `titanic_partitions` won't have been created.

Section 1.3. Create titanic_model

```
# Make sure sas.teach_me_SAS is set to False (just in case).
sas.teach_me_SAS(False)

# Now let's create an object that will allow us to access SAS/STAT procedures.
sas_stat = sas.sasstat()

# We'll also set our response variables and explanatory variables.
outcome = "survived(event='1')"
covariates = 'pclass sex age siblings_spouses_aboard parents_children_aboard fare'

# We're now ready to use PROC LOGISTIC to train a model and score the test dataset.
titanic_model = sas_stat.logistic(
```

```
    data = titanic_partitions[0],
    cls = 'sex',
    model = f'{outcome} = {covariates}',
    stmtpassthrough = f'score data={titanic_partitions[1].table} out=scored fitstat',
    procopts = 'plots=none',
)

# Let's check the SAS log to see how everything went.
display(titanic_model.LOG)

# What else is in this titanic_model object?
print(dir(titanic_model))
```

```
63                                                      The SAS System                    Tuesday, Februa

408        options nosource;
536

64                                                      The SAS System                    Tuesday, Februa

537
```

Concept Check 1.3

- Try this, and see what happens: Change `LOG` on Line 21 to any of the object names in the list that was output by Line 24.
- True or False: The Python object `titanic_model` has 13 sub-objects nested inside of it.
- Fun Fact: The strings `"survived(event='1')"` and `'survived(event="1")'` produce identical behavior in Python.

```
549
```

Solution: True! There are 13 items in the list printed out by Line 24.

```
65                                                      The SAS System                    Tuesday, Februa
```

Section 1.4. Create train_auc and test_auc

```
        'ASSOCIATION'.
```

```
 1 # But how does the model perform?
 2 display(titanic_model.ASSOCIATION)
 3 display(titanic_model.SCOREFITSTAT)
 4
 5 # Let's pull out the AUC value for training.
 6 train_auc_srs = titanic_model.ASSOCIATION.loc[titanic_model.ASSOCIATION['Label2'] == 'c', 'nValue2']
 7 print('\n')
 8 print(type(train_auc_srs))
 9 print('\n')
10 print(train_auc_srs)
11 train_auc = train_auc_srs.iloc[0]
12
13 # Let's also pull out the AUC value for test.
14 test_auc = titanic_model.SCOREFITSTAT['AUC'][0]
15
```

```
16 # And, finally, let's compare them.
17 print('\n')
18 print(f'Training AUC: {train_auc:.4f}, Test AUC: {test_auc:.4f}')
```

	Label1	cValue1	nValue1	Label2	cValue2	nValue2
0	Percent Concordant	86.5	86.532990	Somers' D	0.731	0.730761
1	Percent Discordant	13.5	13.456928	Gamma	0.731	0.730834
2	Percent Tied	0.0	0.010082	Tau-a	0.339	0.338866
3	Pairs	89270	89270.000000	c	0.865	0.865380

	DataSet	Freq	LogLike	MisClass	AUC	AIC	AICC	BIC	SC	RSquare
0	WORK.TITANIC1_SCORE	266.0	-128.118646	0.184211	0.841264	270.237292	270.6714	295.321766	295.321766	0.334099

```
<class 'pandas.core.series.Series'>

3    0.86538
Name: nValue2, dtype: float64

Training AUC: 0.8654, Test AUC: 0.8413
```

Concept Check 1.4

- Short Answer: How would you access specific cells of a dataset in SAS?
- Fun Facts:
  - A DataFrame in Python is a tabular data structure with rows and columns, similar to a SAS data set. The columns of a DataFrame are called `Series`.
  - 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 (like on Line 6 above), but it also means the size of DataFrames can't grow beyond available memory.
  - Problem solving always involves trade-offs. Python provides flexible access to in-memory data, whereas SAS makes it possible to work with datasets whose size exceeds available memory.

Solution: We can subset a SAS dataset to specific rows and columns with KEEP, DROP, and WHERE dataset options, or in a SQL query with SELECT, WHERE, and HAVING clauses. To pull out a single value, we might store it in a macro variable with CALL SYMPUTX in a DATA step, or an INTO expression in PROC SQL.

Section 1.5. Alternate ways to create train_auc

```
# We could simplify getting the training AUC by just hard-coding its row index.
train_auc = titanic_model.ASSOCIATION['nValue2'][3]
print(train_auc)
```

```
0.86538030693402
```

```
# Or we can create an index on the Label2 column and get the best of both worlds.
indexed_association = titanic_model.ASSOCIATION.set_index('Label2')
train_auc = indexed_association['nValue2']['c']
print(train_auc)
```

```
0.86538030693402
```

Concept Check 1.5

- Multiple choice: What do you think is the best way to pull out the AUC value from the ASSOCIATION attribute of `titanic_model`?

  A. Using `loc` to subset on the value of a column.
  B. Using column labels and integer row indices.
  C. Using `set_index` to create a non-integer index.

- Fun Fact: This exercise would seem to contradict the often-quoted aphorism "There should be one-- and preferably only one --obvious way to do it" from the Zen of Python.

Solution: The authors prefers option C (`set_index`), but there are merits to all three methods.

Section 1.6. Additional Exercises

For practice, we recommend the following:

1. Run the code in the cell below (taken from Section 1.4).
2. Modify the code to instead pull the `Percent Concordant` value out of `titanic_model.ASSOCIATION`.
3. Use the `display` method to view the contents of `titanic_model.PARAMETERESTIMATES`.
4. Finally, pull a single value out of `titanic_model.PARAMETERESTIMATES`. (E.g., you might try getting the parameter estimate for the variable `Age`, which you can do using any of the three methods discussed above.)

```
display(titanic_model.ASSOCIATION)
train_auc_srs = titanic_model.ASSOCIATION.loc[titanic_model.ASSOCIATION['Label2'] == 'c', 'nValue2']
print(train_auc_srs)
```

	Label1	cValue1	nValue1	Label2	cValue2	nValue2
0	Percent Concordant	86.5	86.532990	Somers' D	0.731	0.730761
1	Percent Discordant	13.5	13.456928	Gamma	0.731	0.730834
2	Percent Tied	0.0	0.010082	Tau-a	0.339	0.338866
3	Pairs	89270	89270.000000	c	0.865	0.865380

```
3    0.86538
Name: nValue2, dtype: float64
```

```
display(titanic_model.ASSOCIATION)
train_percent_concordant_srs = titanic_model.ASSOCIATION.loc[titanic_model.ASSOCIATION['Label1'] == 'Percent Co
print(train_percent_concordant_srs)

```

```
5 display(titanic_model.PARAMETERESTIMATES)
6 age_parameter_estimate = titanic_model.PARAMETERESTIMATES.loc[titanic_model.PARAMETERESTIMATES['Variable'] == '
7 print(age_parameter_estimate)
```

	Label1	cValue1	nValue1	Label2	cValue2	nValue2
0	Percent Concordant	86.5	86.532990	Somers' D	0.731	0.730761
1	Percent Discordant	13.5	13.456928	Gamma	0.731	0.730834
2	Percent Tied	0.0	0.010082	Tau-a	0.339	0.338866
3	Pairs	89270	89270.000000	c	0.865	0.865380

```
0    86.53299
Name: nValue1, dtype: float64
```

	Variable	ClassVal0	DF	Estimate	StdErr	WaldChiSq	ProbChiSq	_ESTTYPE_
0	Intercept	NaN	1.0	4.284323	0.647637	43.762334	3.707725e-11	MLE
1	Pclass	NaN	1.0	-1.345140	0.180891	55.297173	1.036175e-13	MLE
2	Sex	female	1.0	1.441836	0.125068	132.903631	9.490878e-31	MLE
3	Age	NaN	1.0	-0.044455	0.009402	22.354893	2.266349e-06	MLE
4	Siblings_Spouses_Abo	NaN	1.0	-0.372227	0.137423	7.336604	6.756436e-03	MLE
5	Parents_Children_Abo	NaN	1.0	-0.106334	0.158890	0.447865	5.033507e-01	MLE
6	Fare	NaN	1.0	0.002022	0.003070	0.433949	5.100575e-01	MLE

```
3   -0.044455
Name: Estimate, dtype: float64
```

Notes and Resources

1. For more about the `pandas` package, including the methods used above, see the following:
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html

    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.set_index.html
    - https://pandas.pydata.org/docs/reference/api/pandas.Series.html
2. For more about the `rich` package, see https://rich.readthedocs.io/
3. For more about the `saspy` package, including the methods used above, see the following:
    - 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.obs
    - https://sassoftware.github.io/saspy/api.html#saspy.sasdata.SASdata.partition
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.lastlog
    - https://sassoftware.github.io/saspy/api.html#saspy.SASsession.read_csv
    - https://sassoftware.github.io/saspy/api.html#saspy.sasstat.SASstat
    - https://sassoftware.github.io/saspy/api.html#saspy.sasstat.SASstat.logistic
4. For more information on built-in Python data structures, such as tuples, see https://jakevdp.github.io/WhirlwindTourOfPython/06-built-in-data-structures.html.
5. For more information on f-strings (i.e., Python strings like `f'{outcome} = {covariates}'`), see https://realpython.com/python-f-strings/.
6. We welcome follow-up conversations. You can connect with us on LinkedIn or email us at isaiah.lankham@gmail.com and matthew.t.slaughter@gmail.com
7. 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