Everything is Better with Friends

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

Setup for Part 1

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 5.5 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=1c8815f0177820993517a4d
  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 135

Access Method         = IOM
SAS Config name       = default
SAS Config file       = /usr/local/lib/python3.7/dist-packages/saspy/sascfg.py
WORK Path             = /saswork/SAS_work3160000013E6_odaws03-usw2.oda.sas.com/SAS_workB05A000013E6_odaws03-
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 = 5094
```

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'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
```

```
Collecting rich
  Downloading rich-12.4.1-py3-none-any.whl (231 kB)
     |████████████████████████████████| 231 kB 4.7 MB/s
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in /usr/local/lib/python3.7/dist-packages (from rich)
Collecting commonmark<0.10.0,>=0.9.0
  Downloading commonmark-0.9.1-py2.py3-none-any.whl (51 kB)
     |████████████████████████████████| 51 kB 4.8 MB/s
Requirement already satisfied: typing-extensions<5.0,>=4.0.0 in /usr/local/lib/python3.7/dist-packages (from
Installing collected packages: commonmark, rich
Successfully installed commonmark-0.9.1 rich-12.4.1
```

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/
proc import datafile=x out=work.'titanic'n dbms=csv replace;  run;
```

	Member	Num	Variable	Type	Len	Pos	Format	Informa
0	WORK.TITANIC	5.0	Age	Num	8.0	16.0	BEST12.	BEST3
1	WORK.TITANIC	8.0	Fare	Num	8.0	40.0	BEST12.	BEST3

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.

```
   Survived  Pclass    Name    Sex  Age  Siblings/Spouses Aboard  Parents/
```

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

```
0          0.0       3.0      Harris     male   22.0                          1.0
```

Section 1.2. Create titanic_partitions

```
                              Bradley
```

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

# Let's partition the dataset into subsets for training and testing a predictive model.
titanic_partitions = titanic_sds.partition(seed=42, singleOut=False)

# 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
Dsopts  =
Results = Pandas
, Libref  = work
Table   = titanic1_score
Dsopts  =
Results = Pandas
)

621 observations for training
266 observations for test
```

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

```python
# 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(titanic_model.__dict__['_names'])
```

```
63                                                         The SAS System

408        options nosource;
536

64                                                         The SAS System

537
538        %macro proccall(d);
539        proc logistic data=work.'titanic1_train'n  plot=all plots=none ;
540        class sex;
541        model survived(event='1') = pclass sex age siblings_spouses_aboa
542        score data=titanic1_score out=scored fitstat ;
543        run; quit; %mend;
544        %mangobj(log0001,logistic,'titanic1_train'n);
548
549
550

65                                                         The SAS System

551
[
    'ASSOCIATION',
    'CLASSLEVELINFO'
```

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.

```
    'RESPONSEPROFILE',
    'SCOREFITSTAT'
```

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

Section 1.4. Create train_auc and test_auc

```python
# But how does the model perform?
display(titanic_model.ASSOCIATION)
display(titanic_model.SCOREFITSTAT)

# Let's pull out the AUC value for training.
train_auc_srs = titanic_model.ASSOCIATION.loc[titanic_model.ASSOCIATION['Label2'] == 'c', 'nValue2']
print('\n')
print(type(train_auc_srs))
print('\n')
print(train_auc_srs)
train_auc = train_auc_srs.iloc[0]

# Let's also pull out the AUC value for test.
test_auc = titanic_model.SCOREFITSTAT['AUC'][0]

# And, finally, let's compare them.
print('\n')
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	A
0	WORK.TITANIC1_SCORE	266.0	-128.118646	0.184211	0.841264	270.237292	270.6

```
<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 Fact: A DataFrame in Python is a tabular data structure with rows and columns, similar to a SAS data set. 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 (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_percent_concordant_srs = titanic_model.ASSOCIATION.loc[titanic_model.ASSOCIATION['Label1'] == 'Percent Co
print(train_percent_concordant_srs)

display(titanic_model.PARAMETERESTIMATES)
age_parameter_estimate = titanic_model.PARAMETERESTIMATES.loc[titanic_model.PARAMETERESTIMATES['Variable'] == '
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
0	Intercept	NaN	1.0	4.284323	0.647637	43.762334	3.707725e 11
1	Pclass	NaN	1.0	-1.345140	0.180891	55.297173	1.036175e 13
2	Sex	female	1.0	1.441836	0.125068	132.903631	9.490878e 31
3	Age	NaN	1.0	-0.044455	0.009402	22.354893	2.266349e 06
4	Siblings_Spouses_Abo	NaN	1.0	-0.372227	0.137423	7.336604	6.756436e

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