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. 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 is
11     iomhost = ['odaws01-usw2.oda.sas.com', 'odaws02-usw2.oda.sas.com', 'odaws03-usw
12 
13     # The following line should be uncommented if, and only if, your ODA account is
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
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 6.9 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: matthew.t.slaughter@kpchr.org
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_workD855000101EF_odaws02-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 = 66031
```

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

# pprint is useful for displaying complex data structures
from pprint import pprint

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

```
Collecting rich
  Downloading rich-10.15.2-py3-none-any.whl (214 kB)
     |████████████████████████████████| 214 kB 7.5 MB/s
Collecting commonmark<0.10.0,>=0.9.0
  Downloading commonmark-0.9.1-py2.py3-none-any.whl (51 kB)
     |████████████████████████████████| 51 kB 8.0 MB/s
Requirement already satisfied: typing-extensions<5.0,>=3.7.4 in /usr/local/lib/py
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)
Installing collected packages: commonmark, colorama, rich
Successfully installed colorama-0.4.4 commonmark-0.9.1 rich-10.15.2
```

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/tit
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?
sas.teach_me_SAS(True)
sas.read_csv(file=titanic_url,table='titanic',libref='work')
sas.teach_me_SAS(False)

# 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

filename x url "https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff
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.
7	WORK.TITANIC	1.0	Survived	Num	8.0	0.0	BEST12.	BEST32.

Concept Check 1.1

- True or False: The SASdata object `titanic_sds` is a data structure kept in memory in our local Google Colab session.

- *False. `titanic_sds` is a pointer to a SAS dataset stored on disk in the remote SAS ODA session.*

Section 1.2. Create titanic_partitions

Mr. Owen

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

# And then print some information about each subset.
print(type(titanic_partitions))
print('\n')
pprint(titanic_partitions)
print('\n')
print(f'{titanic_partitions[0].obs()} observations for training')
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

- True or False: A `tuple` is equivalent to a `list` in Python.

- *False. A `tuple` differs from a list in that it's of fixed length and immutable. meaning its values can't be changed, whereas both the length and values in a `list` are allowed to change freely.*

Section 1.3. Create titanic_model

```
# 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_aboard pa
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
```

Concept Check 1.3

- True or False: `"survived(event='1')"` and `'survived(event="1")'` produce identical behavior.

```
    'FITSTATISTICS',
    'GLOBALTESTS'
```

- *True. Double quotes and single quotes always produce the same behavior in Python.*

```
    'NOBS',
```

Section 1.4. Create train_auc and test_auc

```
    'SCOREFITSTAT',
```

```
# 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'] =
print('\n')
print(type(train_auc_srs))
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	AICC	
0	WORK.TITANIC1_SCORE	266.0	-128.118646	0.184211	0.841264	270.237292	270.6714	2

```
<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?

- *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)
```

```
0.8653803069340201
```

```
# 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)
```

```
0.8653803069340201
```

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.
- *The author 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 cell below to list and display the attributes of `titanic_model`.
2. Pick an output object from `titanic_model` and use loc, iloc, and/or set_index to pull out subsets or specific values.

```
print(titanic_model.__dict__['_names'])
titanic_model.ALL()
```

```
[
    'ASSOCIATION',
    'CLASSLEVELINFO',
    'CONVERGENCESTATUS',
    'FITSTATISTICS',
    'GLOBALTESTS',
    'MODELANOVA',
    'MODELINFO',
    'NOBS',
    'ODDSRATIOS',
    'PARAMETERESTIMATES',
    'RESPONSEPROFILE',
    'SCOREFITSTAT',
    'LOG'
]
```

	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

	Class	control_var	Value	X1
0	Sex	0	female	1.0
1	NaN	0	male	-1.0

```
# The solution will vary based on which output objects are chosen.

# One option, shown here, is to pull the parameter estimates and odds ratios for a
display(titanic_model.ODDSRATIOS)
display(titanic_model.PARAMETERESTIMATES)

# Let's pull out the odds ratio for Age.
indexed_odds_ratios = titanic_model.ODDSRATIOS.set_index('Effect')
age_odds_ratio = indexed_odds_ratios['OddsRatioEst']['Age']

# Let's also pull out the parameter estimate for Age.
indexed_paramter_estimates = titanic_model.PARAMETERESTIMATES.set_index('Variable')
age_paramter_estimate = indexed_paramter_estimates['Estimate']['Age']

# And, finally, let's compare them.
print('\n')
print(f'OR for Age: {age_odds_ratio:.4f}, Paramter Estimate for Age: {age_paramter_
```

	Effect	OddsRatioEst	LowerCL	UpperCL
0	Pclass	0.260503	0.182742	0.371353
1	Sex female vs male	17.879809	10.950807	29.193061
2	Age	0.956518	0.939053	0.974309
3	Siblings_Spouses_Abo	0.689197	0.526464	0.902233
4	Parents_Children_Abo	0.899125	0.658525	1.227630
5	Fare	1.002024	0.996013	1.008071

	Variable	ClassVal0	DF	Estimate	StdErr	WaldChiSq	ProbChiSq	_ES
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-	

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 `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

3. For more information on built-in Python data structures, such as tuples, see https://jakevdp.github.io/WhirlwindTourOfPython/06-built-in-data-structures.html.
4. For more information on f-strings (i.e., Python strings like `f'{outcome} = {covariates}'`), see https://realpython.com/python-f-strings/.
5. 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
6. 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