Paper 3610-2019

SWAT's it all about? SAS Viya® for Python Users

Carrie Foreman, Amadeus Software Limited

ABSTRACT

Python® is described within the community as the "second-best language for everything" (CALLAHAN, 2018). The strength of Python as a language stems from its huge range of uses including as a tool for data science. Python is taught on many university courses and as a result there are a vast number of coders with Python skills in the data science industry. Last year at Amadeus we explored the use of SASpy as an initial tool to bridge the gap between Python and SAS 9.4 (FOREMAN, 2018), this year we move on to look at integration with SAS Viya®.

In July 2016, SAS® released Python SWAT (Scripting Wrapper for Analytics Transfer), a Python library. It allows connections to CAS (Cloud Analytic Services) and therefore opens up SAS Viya's functionality to Python users. SWAT allows users who do not have a SAS background to perform their data manipulation and analytics using more familiar Pythonic syntax whilst still harnessing the power of CAS and SAS Viya.

In this paper we will demonstrate the use of Python SWAT to first connect to a CAS session, load data from Python into CAS and use CAS action sets to analyze it. We will then make the data available to other applications such as SAS Visual Analytics to demonstrate some of the functionality this gives.

INTRODUCTION

Python is an integral part of the data science culture and is used within many organizations worldwide. Many universities teach Python as part of technology courses creating a steady stream of Python enabled programmers who then take those skills into industry. Python-SWAT provides these Python users more options for developing data science solutions. Now, these users can not only create models within Python interfaces, but they can also use the features made available with SAS Viya to complete this.

SAS Viya architecture is designed to work with data that is loaded into memory. Python-SWAT allows Python users to connect to this data using an interface that they are familiar with. This single data store, enables a centralization of governance for the platform whilst enabling users to choose their own tools. Python users can also exploit the features available within SAS Viya and tools such as SAS Visual Analytics to create visualizations of their data and data science models.

The connectivity provided by SWAT allows both Python and SAS users to collaborate on a platform that is centralized and scalable. Python users and SAS users can now work side-by-side completing data manipulation, data visualization and data modelling in the language of their choice.

PYTHON

Originally named after the BBC series "Monty Python's Flying Circus", Python is an open source object-oriented programming language which was first released in 1991. The language is currently used by millions of individuals worldwide for data science.

JUPYTERLAB

Jupyter is an open source programming interface which encompasses three main programming languages; Julia, Python and R into a single platform. The original version of

Jupyter named the Jupyter Notebook was first developed in 2014 and has evolved into the latest version of JupyterLab which was released in early 2018. The code developed within this paper can be submitted within both evolutions; Jupyter Notebook and JupyterLab. To install the latest version of JupyterLab, the Anaconda distribution is available, which includes Python and any dependencies which are installed using a wizard (Anaconda, 2019).

PYTHON PACKAGES

Python is not only used for data science, but also web development, application development and video game design. With a language that is so versatile it is important to note that not all its functionality is included within the Anaconda distribution which was created specifically for data science. To limit the size of the Python platform, any additional functionality can be added by installing additional packages, also referred to as libraries, to the Python distribution. This enables users to install the functionality that they require by adding only the relevant packages to their distribution. A Python Package Index (also known as the *Cheese Shop*) is available at:

```
https://pypi.org/
```

The Python Package Index details all the packages which are currently available to download and is accessible from the Python website (Python, 2019).

SWAT

The SAS Scripting Wrapper for Analytics Transfer (SWAT) is a package developed by SAS to allow the Python interface to access the SAS Viya platform. The package allows connections to be made to the SAS Cloud Analytic Services (CAS) engine to enable in-memory processing for Python users in an environment that is familiar to them. It is also now possible to load data directly from a Python data frame into CAS, to allow further visualizations to be completed in Python, SAS Studio or SAS Visual Analytics. SWAT enables both Python and SAS users to collaborate on a centralized platform.

Required Packages

To allow connections to SAS Viya through SWAT, the following additional packages are required to be installed via pip as described within Table 1:

Package	Description	Install command
pip	Pip is a Python package manager used to install any additional packages.	`easy_install pip`
pandas	Pandas is a Python library which was developed for data manipulation and analysis. The SWAT package has dependencies on Pandas.	`pip install pandas`
Matplotlib	Matplotlib is a Python 2D plotting library used to create visualizations such as box plots and scatter plots. The SWAT package has dependencies on matplotlib to view any visualizations created.	`pip install matplotlib`
SWAT	SWAT is a package created by SAS to allow Python to connect to CAS (Cloud Analytics Services). The library allows users to load data into CAS increasing the processing power which they can utilize and allowing them to code in an environment they are familiar with.	`pip install swat`

Table 1: Packages required for SWAT

Note: pip, pandas and matplotlib are installed as part of the Anaconda distribution of Python.

Importing Packages into Python

Although the packages have been installed, these are not automatically available within Python. To enable a Python session to use these packages they need to be imported. We can import the required packages for SWAT using the keyword **import** and the name of the package to import. For some packages it can be beneficial to use an alias to call the package with the **as** keyword. We can import the key packages as below:

```
import swat
import pandas as pd
import matplotlib.pyplot as plt
```

CONNECTING TO A CAS SESSION

Once required libraries have been imported, it is now possible to make a connection to SAS Viya. A connection statement specifies a host, port, user and password for a CAS session and creates a session object. Two options are possible for a connection statement as below.

Option 1: Use Python environment variables to prevent plain text passwords being displayed within code. Each of these Python variables can be set within the SWAT configuration:

```
sess = swat.CAS(cashost, casport, user, password)
```

Option 2: For a quick test, the values can be directly typed into the connection statement:

```
sess = swat.CAS('sas.server.com', 5770, 'myuser', 'mypassword')
```

On submitting the connection statement, it is not immediately clear that the code has run successfully, as Jupyter does not display any output. To verify that a connection is now made, the **serverstatus** CAS action can be run on the CAS session to display details about the current SAS Viya session. The action is submitted as follows:

```
sess.serverstatus()
```

Output 1 details the default output expected to be displayed within the Jupyter Notebook below the code cell after a successful connection has been performed.

```
NOTE: Grid node action status report: 1 nodes, 8 total actions executed.
```

§ About

{'CAS': 'Cloud Analytic Services', 'Version': '3.04', 'VersionLong': 'V.03.04M0P07112018', 'Copyright': 'Copyright © 2014-2018 SAS Institute Inc. All Rights Reserved.', 'ServerTime': '2019-02-14T14:14:01Z', 'System': {'Hostname': 'sasserver', 'OS Name': 'Linux', 'OS Family': 'LIN X64', 'OS Release': '2.6.32-696.18.7.el6.x86_64', 'OS Version': '#1 SMP Thu Jan 4 17:31:22 UTC 2018', 'Model Number': 'x86_64', 'Linux Distribution': 'CentOS release 6.9 (Final)'}, 'license': {'site': 'DEMOCENTER EEC-TRIAL-FULL 8.3.1', 'siteNum': 70180938, 'expires': '11Oct2019:00:00:00', 'gracePeriod': 45, 'warningPeriod': 45, 'maxCPUs': 9999}}

§ server

Server Status

	nodes	actions
0	1	8

§ nodestatus

Node Status

	name	role	uptime	running	stalled
0	sasserver.demo.sas.com	controller	4.947	0	0

elapsed 0.00151s · sys 0.001s · mem 0.295MB

Output 1: Confirming a connection to CAS

If a connection attempt is not successful, Output 2 may be displayed below the code cell.

```
---------------------------------------------------------------------------
SWATError                                 Traceback (most recent call last)
<ipython-input-135-cc1df86559c1> in <module>()
      1 # Load data into a local CASLIB
      2 castbl = sess.upload_file(data='/home/carrie.foreman/Jupyter_Saved_Work/StudentsPerformance.csv',
----> 3                           casout=dict(name='StudentsPerformance', caslib='casuser'))

/opt/sasinside/anaconda3/lib/python3.6/site-packages/swat/cas/connection.py in upload_file(self, data, importoptions, casout, *
*kwargs)
   1460
   1461         if out.severity > 1:
-> 1462             raise SWATError(out.status)
   1463
   1464         return out['casTable']

SWATError: The action was not successful.
```

Output 2: An unsuccessful attempt at loading data into CAS.

CAS Action Sets

SAS Viya provides access to CAS actions. CAS actions are tools which have been developed to perform a particular task. All CAS actions are grouped into bundles of CAS actions with common functionality called CAS action sets. Action sets are available within SAS Viya to complete tasks such as:

- Modify access controls
- Submit DS2 code
- Submit FedSQL code
- Complete machine learning
- Complete text mining

The SAS documentation contains a complete list of the available action sets dependent on the SAS products licensed:

https://documentation.sas.com/?docsetId=pgmdiff&docsetTarget=p06ibhzb2bklaon1a86ili3wpil9.htm&docsetVersion=3.3&locale=en

SWAT enables Python users to gain access to some pre-loaded CAS action sets. It is also possible to load additional action sets into the Python session to gain access to their functionality. The '**simple'** CAS action set gives access to the Python methods summary, frequencies and crosstabs. To load the simple action set into the Python session the following **loadactionset** method can be run:

```
sess.loadactionset('simple')
```

To confirm that the simple CAS action set has been added Output 3 should be displayed:

```
NOTE: Added action set 'simple'.

§ actionset
simple

elapsed 0.000343s · mem 0.194MB
```

Output 3 displays that the 'simple' CAS action set has been loaded into the Python session.

CONNECTING TO DATA

In this paper we will import data from a CSV (Comma-Separated Values) file. The data contains information on a class of students' performance detailing results for their math, reading and writing exams.

Table 2 contains information on the variables included within the Student Performance table.

Variable Name	Description
gender	male or female
race/ethnicity	Groups A to E
parental level of education	Details the level of education a student's parents hold from the following categories: • associate's degree • bachelor's degree • high school • master's degree • some college • some high school
lunch	Identifies if a student is entitled to a free or reduced lunch cost or the standard charges apply for lunch
test preparation course	completed or none
math score	Score out of 100
Reading Score	Score out of 100
Writing Score	Score out of 100

Table 2: Data dictionary for the Students Performance table.

LOADING DATA INTO CAS

To load data into CAS, Python users have two options which they can exploit that do not require any SAS coding knowledge. They can use Python code, with the help of SWAT, which provides several methods for uploading files into CAS.

Alternatively, Python users can use the SAS Viya interface provided through SAS Data Preparation. This provides a simple point-and-click interface where files can be drag-and-dropped to be loaded into CAS.

Note: It is also possible for SAS users to load data within SAS Studio using SAS code or CAS actions.

This paper focuses on the Python coding capabilities for loading data into CAS. To upload data using Python coding we will utilize the **upload_file** method which calls the upload_file CAS action (SAS, 2016). This can be used to read in the data from the CSV file and load it directly into CAS. This method was chosen as it is flexible enough to be used with multiple file types.

Table 3 details the file types which can be uploaded using the upload_file method:

BASESAS	ESP	HDAT	SPSS
CSV	EXCEL	JMP	XLS
DTA	FMT	LASR	

Table 3: File types which can be loaded using the upload_file method

To load the CSV file from a location on the SAS Viya server, the following statement is run using the **upload_file** method:

```
castbl = sess.upload_file(data='/home/carrie.foreman/Students.csv',
                          casout=dict(name='StudentsPerformance',
                          caslib='casuser', replace=True))
```

The **casout** option has been used to present the following additional options which are defined within a Python dictionary:

- name: Specifies the name of the CAS table once it is uploaded into CAS.
- caslib: The CAS library which the CAS table will be uploaded into. The casuser library is a library which will make the data set available to the current user only.
- replace: Used to force a data set to be replaced within CAS, if this is set to False and a CAS table already exists this will generate an error.

If the CSV has been uploaded successfully similar information to that displayed within Output 4: Confirming that a file has been uploaded into CAS. should be displayed below the code cell.

```
NOTE: Cloud Analytic Services made the uploaded file available as table STUDENTSPERFORMANCE in caslib CASUSER(carrie.foreman@am
adeus.co.uk).
NOTE: The table STUDENTSPERFORMANCE has been created in caslib CASUSER(carrie.foreman@amadeus.co.uk) from binary data uploaded
to Cloud Analytic Services.
```

Output 4: Confirming that a file has been uploaded into CAS.

Note: Jupyter will report an error below the code cell if any issues are encountered when the data is uploaded.

VIEWING THE DATA

The data can now be called using the castbl Python variable. To view the data that has been loaded, the **head** method can be used, which by default displays only the first five observations of the data set. It is possible to increase this by using a number within the brackets of the method as shown below.

```
castbl.head(10)
```

Output 5 shows the results of the head method displaying the first 8 observations within the CAS table, confirming the **studentsperformance** table is currently available within CAS.

Selected Rows from Table STUDENTSPERFORMANCE

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	F	group B	bachelor's degree	standard	0	72.0	72.0	74.0
1	F	group C	some college	standard	1	69.0	90.0	88.0
2	F	group B	master's degree	standard	0	90.0	95.0	NaN
3	M	group A	associate's degree	free/reduced	0	47.0	57.0	44.0
4	M	group C	some college	standard	0	76.0	78.0	75.0
5	F	group B	associate's degree	standard	0	71.0	83.0	78.0
6	F	group B	some college	standard	1	88.0	95.0	92.0
7	M	group B	some college	free/reduced	0	40.0	43.0	39.0
8	M	group D	high school	free/reduced	1	64.0	64.0	67.0
9	F	group B	high school	free/reduced	0	38.0	60.0	50.0

Output 5: Viewing data loaded into CAS.

DATA EXPLORATION

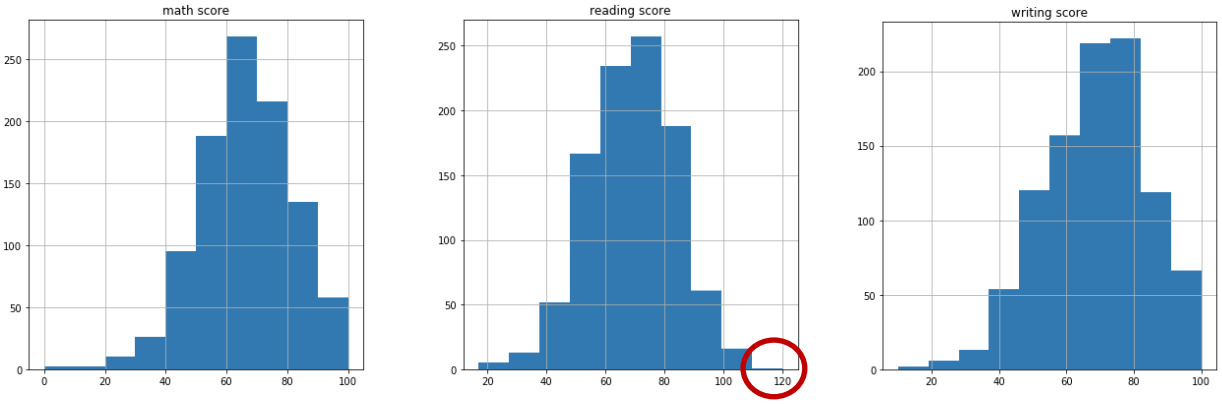
To perform an initial data exploration, Python contains many packages and libraries with different capabilities. In this section, SWAT will be used to explore the data. SWAT can be used to create visualizations of the data to view any distributions which may be present. To view the plots which are created with SWAT the matplotlib library needs to be imported into the Python session.

HISTOGRAMS

Histograms are a great way to view the distribution of any numeric variables within a data set. They can be used to identify any issues or outliers in the data. To create a histogram of all numeric variables within the CAS table the Python **hist** method can be used. The hist method can be applied to the table using the following code:

```
castbl.hist(figsize=(10,10))
plt.show()
```

This hist method produces one histogram for each numeric variable within the data set. The figsize option included within the code specifies the size of the plot which you would like to be created. For the students' data set, one histogram is created for the scores for the math, reading and writing exams. Output 6 displays three plots which are generated for the **studentsperformance** CAS table:



Output 6: Histograms of all numeric variables within the CAS table.

Viewing the three histograms which have been created, all three plots appear slightly negatively skewed. The math score and writing score plots appear to display no obvious outliers, however the reading score appears to show a maximum value around 120 as highlighted in Output 6. Considering the value is a percentage it would be impossible to achieve 120% on a reading test. It would now be advisable for the programmer to introduce data quality checks to address this erroneous data.

IDENTIFYING MISSING VALUES

Missing values can cause issues with some data visualization techniques or data science models. To prevent this, it is important that a programmer addresses these issues during the data manipulation stage. Within Python, the info method can be used to provide information on the CAS table. This includes information on whether missing values are present for each variable. To access this information the **info** method can be applied to the CAS table as below:

```
castbl.info()
```

The results of running the **info** method on the CAS table are displayed within Output 7.

```
CASTable('STUDENTSPERFORMANCE', caslib='CASUSER(carrie.foreman@amadeus.co.uk)')
Data columns (total 8 columns):
                              N   Miss     Type
gender                     1000  False  varchar
race/ethnicity             1000  False  varchar
parental level of education  1000  False  varchar
lunch                      1000  False  varchar
test preparation course    1000  False  varchar
math score                 1000  False   double
reading score               994   True   double
writing score               978   True   double
dtypes: double(3), varchar(5)
data size: 137039
vardata size: 33039
memory usage: 137072
```

Output 7: Information about the currently loaded CAS table.

The info method displays information on the currently loaded CAS table. This includes the name of the uploaded **studentsperformance** table and the variable names which are present within the CAS table. In addition to this the type of each variable and information on missing values is included.

Looking specifically at the "Miss" column in the info output, the reading score and writing score have values of "True" indicating that missing values are present for these two columns. In addition to this, the "N" column identifies the number of non-missing values that are present in the CAS table. We have identified seven missing values for reading score and twelve missing values for writing score.

DATA MANIPULATION

Once the data has been explored and issues have been identified, we need to rectify these. This can be completed during the data manipulation stage. This section will focus on completing the data manipulation steps within Python, however, the SAS Viya Data Preparation web application can also be used by Python users to create the same results within a point-and-click interface. Additionally, it is also possible for SAS users to manipulate the CAS table using SAS Studio.

IDENTIFYING INCORRECT DATA

In the exploration stage, we identified than an individual had scored approximately 120 in their reading exam. We first need to identify the observation where this score has been recorded. To identify this the **head** method can be used on a subset of the CAS Table. The following code filters the **studentsperformance** CAS table to identify any reading score that is above 100.

```
castbl[castbl['reading score']>100].head()
```

Output 8 displays the results of running the filter of the CAS table.

Selected Rows from Table STUDENTSPERFORMANCE

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	F	group B	master's degree	free/reduced	1	77.0	120.0	94.0

Output 8: Viewing individuals who achieved greater than 100% in their reading test.

A single observation is identified within the CAS table which has a reading score of 120.

ADDRESSING INCORRECT DATA

Once the observation has been identified, we need to decide what we would like to do with this outlier. There are many ways of dealing with outliers including replacing the value or removing the observation. For this record we will temporarily replace the value of the reading score to a null value, which will be handled in the next section.

To complete this the replace CAS action is called using the **replace** method on the CAS table. This can be used to replace any value of 120 in the data set with the value of nan which indicates a Python missing value. The inplace option set to True replaces the value directly in the data set.

```
castbl['reading score'].replace(120, pd.np.nan, inplace=True)
```

Output 9 displays the view within Jupyter after running the replace method.

```
CASColumn('STUDENTSPERFORMANCE', caslib='CASUSER(carrie.foreman@amadeus.co.uk)')['reading score']
```

Output 9: Default output from replacing a value in a CAS table.

The output displays that the **studentsperformance** CAS table has been affected, specifically affecting the reading score column.

To verify that no further values above 100 are present within the CAS table, the **head** method can be re-run on the subset of the data as below:

```
castbl[castbl['reading score']>100].head()
```

Output 10 should be displayed to indicate that no rows have been found:

Selected Rows from Table STUDENTSPERFORMANCE

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score

Output 10: Checking that no students have now achieved more than 100% in their reading test.

As no observations are identified using the head method with the filter, we can confirm that the reading score has been removed from the necessary observation.

IMPUTING MISSING VALUES

Many Python methods are available to remove, replace or impute missing values within the data set. In the exploration section we identified seven missing values for reading score and twelve missing values for writing score. We also have an additional missing value which was created using the replace method to replace the incorrect value of 120 for the reading score.

In this example, we would like to impute any missing values within the CAS table using the median value. To complete this using Python code the pandas **fillna** method can be used:

```
castbl.fillna(castbl.median(), inplace=True)
```

Any missing values within the data set will be replaced with the median of that variable. The additional option of "inplace=True" allows the data set to be overwritten with the new data. Jupyter does not produce any output when completing this command. To verify that this has taken effect we can run the info method on the CAS table:

```
castbl.info()
```

Output 11 displays the results of re-running the info command.

```
CASTable('STUDENTSPERFORMANCE', caslib='CASUSER(carrie.foreman@amadeus.co.uk)')[['gender', 'race/ethnicity', 'parental level of
education', 'lunch', 'test preparation course', 'math score', 'reading score', 'writing score']]
Data columns (total 8 columns):
                              N   Miss     Type
gender                     1000  False  varchar
race/ethnicity             1000  False  varchar
parental level of education  1000  False  varchar
lunch                      1000  False  varchar
test preparation course    1000  False  varchar
math score                 1000  False   double
reading score              1000  False   double
writing score              1000  False   double
dtypes: double(3), varchar(5)
data size: 145039
vardata size: 33039
memory usage: 145072
```

Output 11: Confirming that all missing values have been filled.

The info method shows that there are no longer any missing values for reading score or writing score.

COMBINING CATEGORIES

The parental level of education variable contains six categories. The users have identified that the category "Some high school" was a mistake and should be combined with the "High school" category. To view a list of the categories which are included within the CAS table the **summary** method can be used. Adding a **groupby** option allows us to view statistics for each category which is present within the CAS table. The following code can be used to create a frequency table displaying the number of observations within each category:

```
castbl['math score'].groupby(['parental level of education']
).summary(subset=['n']).concat_bygroups()
```

The results of the summary method are displayed in Output 12.

§ Summary

Descriptive Statistics for STUDENTSPERFORMANCE

	Column	N
parental level of education		
associate's degree	math score	222.0
bachelor's degree	math score	118.0
high school	math score	196.0
master's degree	math score	59.0
some college	math score	226.0
some high school	math score	179.0

Output 12: Categories within the CAS table for Parental Level of Education

The output shows that 196 students have a parent categorized as 'high school', and 179 students have 'some high school' which need to be combined. This can be completed within Python code using a **replace** method on the column parental level of education. The code below replaces the value of 'some high school' in the data set with the value 'high school':

```
castbl['parental level of education'].replace('some high school', 'high
school', inplace=True)
```

The code produces the display in Output 13 below the code cell within Jupyter. This details the column name that has been affected by the code.

```
CASColumn('STUDENTSPERFORMANCE', caslib='CASUSER(carrie.foreman@amadeus.co.uk)')['parental level of education']
```

Output 13: Removing a category from the CAS table and combining.

To confirm that the category has been removed the summary code can be re-run as below:

```
castbl['math score'].groupby(['parental level of education']
).summary(subset=['n']).concat_bygroups()
```

Output 14 details the results from the summary.

§ Summary

Descriptive Statistics for STUDENTSPERFORMANCE

	Column	N
parental level of education		
associate's degree	math score	222.0
bachelor's degree	math score	118.0
high school	math score	375.0
master's degree	math score	59.0
some college	math score	226.0

Output 14: Displaying the number of students with each parental level of education.

The 'some high school' category has now been removed and all 375 students have been reassigned to the 'high school' category.

DATA PROMOTION

Once the data manipulation is completed we can promote the data to the public CAS library. This allows the data to be accessible by all users who have an active CAS session, including users within SAS Studio and SAS Viya web applications as well as those within Python.

PROMOTING DATA FROM A PRIVATE CAS LIBRARY

To promote the **studentsperformance** data set from the casuser library into the public library the **promote** CAS action can be called using the promote method within Python:

```
sess.promote(name=castbl, targetlib='public', target='STUDENTMODIFIED')
```

The promote method requires three parameters including the following:

- name – The Python object which requires promoting to an alternative CAS library.
- targetlib – The CAS library which the CAS table will be promoted to.
- target – The name of the new CAS table within the target library.

Output 15 shows the output from the promote command which details the time the action took to complete:

```
elapsed 0.00186s · user 0.001s · mem 0.269MB
```

Output 15: Promoting a CAS table into the public CAS library.

Note: If the data is unable to be promoted, an error will be displayed below the code cell.

VIEWING DATA AVAILABLE WITHIN A CAS LIBRARY

The **fileinfo** method can be used to identify a list of tables which are currently available in a CAS library. This method lists information about the files which are available including the permissions, owner, group and size of each file. To list all the files within the public CAS library available to all SAS Viya users the following code can be run:

```
sess.fileinfo(caslib='public')
```

A subset of the output which is generated by the fileinfo method is shown within Output 16.

§ FileInfo

	Permission	Owner	Group	Name	Size	Encryption	Time	ModTime
0	-rwxr-xr-x	cas	sas	dispersionzicmp_va.sashdat	103008	NONE	2018-07-18T21:16:59-04:00	1.847582e+09
1	-rwxr-xr-x	cas	sas	USCOUNTYSHAPEFILES.sashdat	13612856	NONE	2018-07-18T21:17:41-04:00	1.847582e+09
2	-rwxr-xr-x	cas	sas	donations.sas7bdat	9568256		2018-07-18T21:16:59-04:00	1.847582e+09
3	-rwxr-xr-x	cas	sas	MINIMUM.sashdat	8752	NONE	2018-07-18T21:17:14-04:00	1.847582e+09
4	-rwxr-xr-x	cas	sas	articles_va.sashdat	59608	NONE	2018-07-18T21:16:56-04:00	1.847582e+09

Output 16 shows a subset of the output of the fileinfo method on the public CAS library.

CONNECTING TO A CAS TABLE WITHIN A DIFFERENT CAS LIBRARY

To work on a CAS table within Python that is available from another CAS library a **CASTable** statement needs to be run. To connect to the **studentmodified** CAS table which was previously promoted, the following CASTable statement can be run:

```
casstumod = sess.CASTable(name='STUDENTMODIFIED', caslib='public')
```

The CASTable method requires two parameters:

- name - The name of the CAS table to connect to.
- caslib – The CAS library which the CAS table is stored.

This allows us to use the **casstumod** Python object to reference the public CAS Table. To verify that the connection is successful, the info method displays the CAS table name and the CAS library for the CAS table:

```
casstumod.info()
```

The results from the info method are displayed in Output 17.

```
CASTable('STUDENTMODIFIED', caslib='public')
Data columns (total 9 columns):
                               N   Miss     Type
gender                      1000  False  varchar
race/ethnicity              1000  False  varchar
parental level of education 1000  False  varchar
lunch                       1000  False  varchar
test preparation course     1000  False  varchar
math score                  1000  False   double
reading score               1000  False   double
writing score               1000  False   double
```

Output 17 the info method, displaying the data within the public CAS library.

DATA VISUALIZATION

Once the data is loaded into CAS, multiple visualization options are available based on the users' preference. Python users can continue to access the data directly through SWAT and continue their coding in Jupyter utilizing packages that they are familiar with. Other users might choose to use the SAS Visual Analytics interface which allows them to drag and drop visualizations into reports without any SAS code knowledge requirements. Additionally, the data is also available for SAS Studio users using SAS code, or custom tasks to create visualizations.

PYTHON VISUALIZATIONS

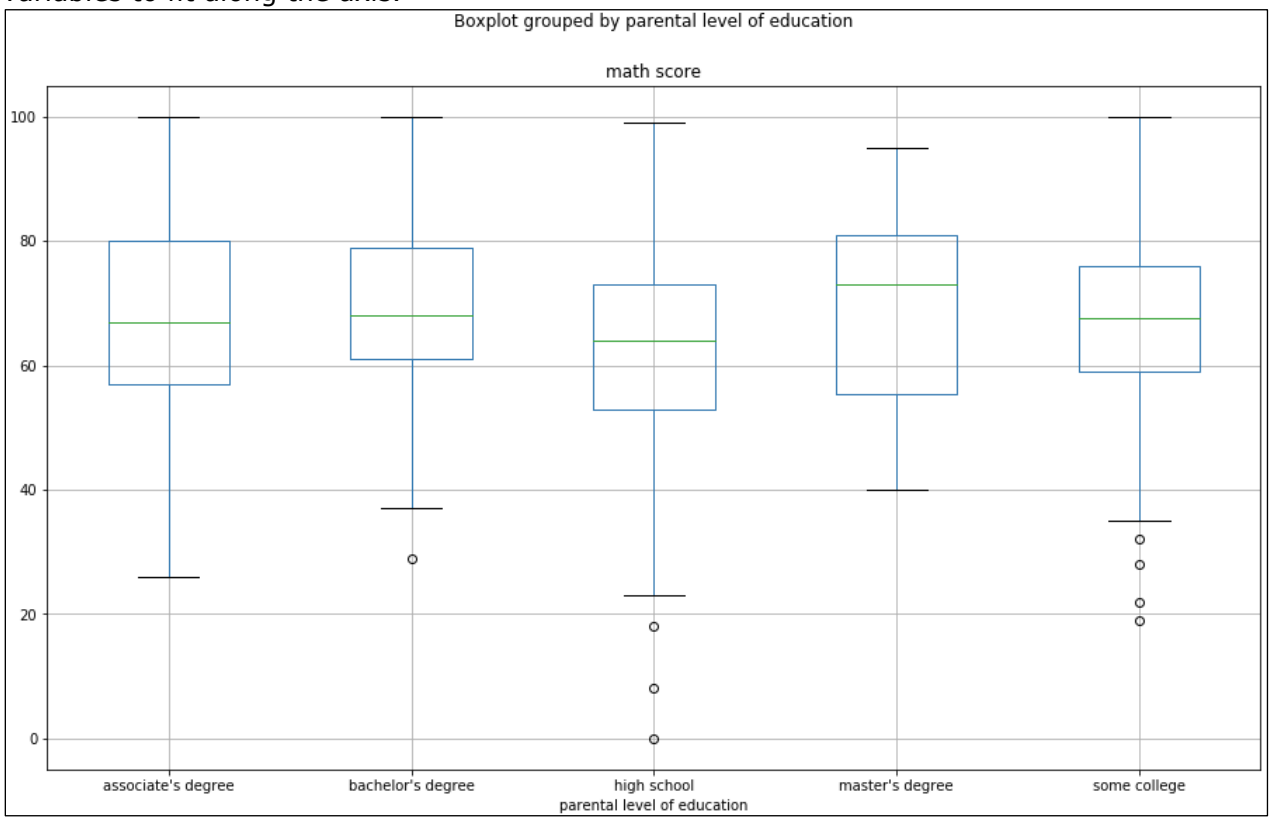
Python contains many methods to create visualizations and models using Python packages. All visualizations within this section are completed using CAS actions which are available through SWAT and matplotlib functionality.

Box Plot

To create a box plot within Python the **boxplot** method can be used on the public CAS table. This has a required argument to provide the column which is to be used for the analysis. For this example, a by statement is also used to group the data by the parental level of education. The code to complete a simple box plot is as follows:

```
casstumod.boxplot(column='math score', by='parental level of education',
                  figsize=(15,9))
plt.show()
```

Output 18 displays the default box plot coloring with an increased figure size to allow the variables to fit along the axis.



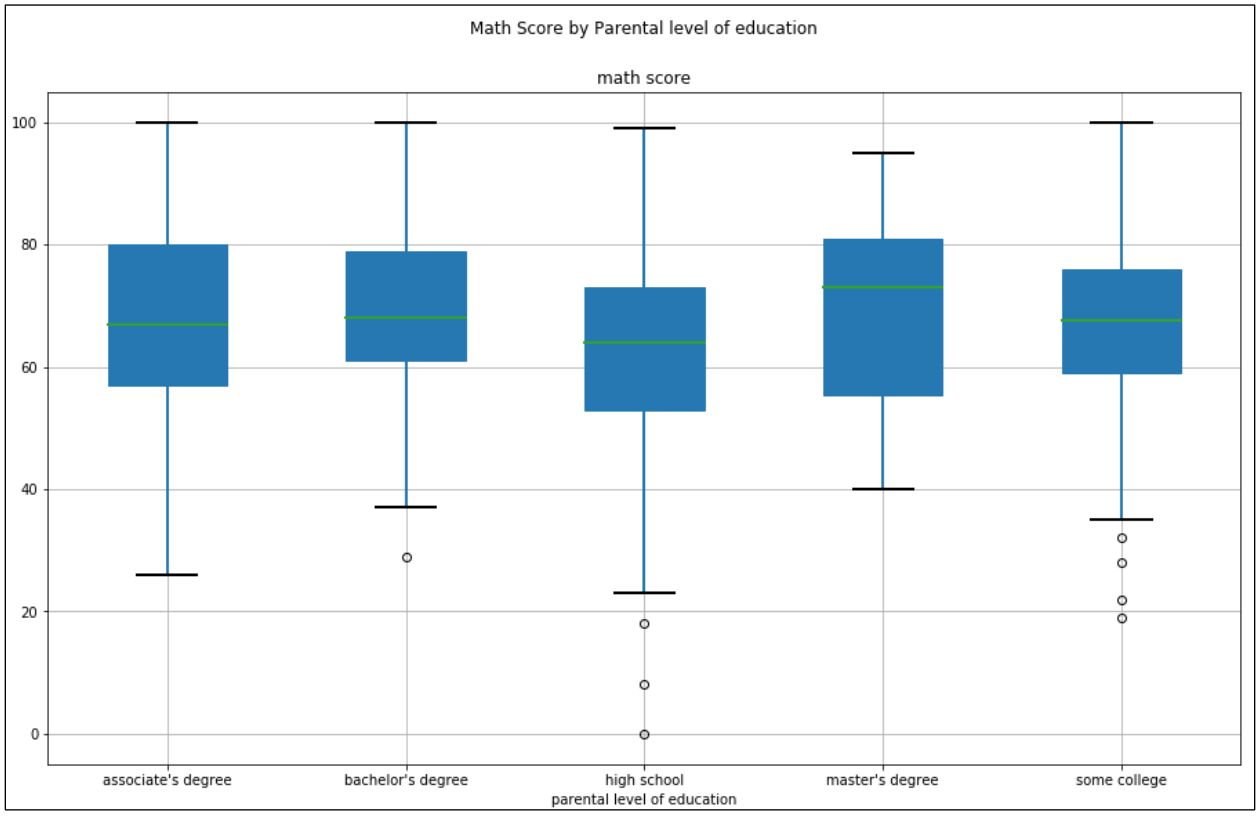
Output 18: The default box plot created within Python

As the default box plots created are not the most visually appealing, a range of options are available to Python users. Options can be added using **patch_artist**, which allows users to modify the coloring and line widths of the box plots. In the following example the linewidth has been increased. Including the **patch_artist** option fills the background of the box in, making the box plot stand out more from the axis. It is also possible to add a custom title to the chart using the **suptitle** option.

The full code to complete this is as below:

```
casstumod.boxplot(column='math score', by='parental level of education',
                  patch_artist=True, cappprops=dict(linewidth=2),
                  whiskerprops=dict(linewidth=2),
                  medianprops=dict(linewidth=2), figsize=(15,9))
plt.suptitle('Math Score by Parental level of education')
plt.show()
```

Output 19 displays the results of the code which are displayed below the code cell.



Output 19: A customized box plot created within Python.

The box plots display the spread of the results for the math exams. It is clear from the plots that those whose parents have a master's degree have higher average math results. In addition to this, those whose parents have a high school education are likely to score lower. From the graph it can also be seen that there are also some outliers for three of the categories:

- High School
- Some College
- Bachelor's Degree

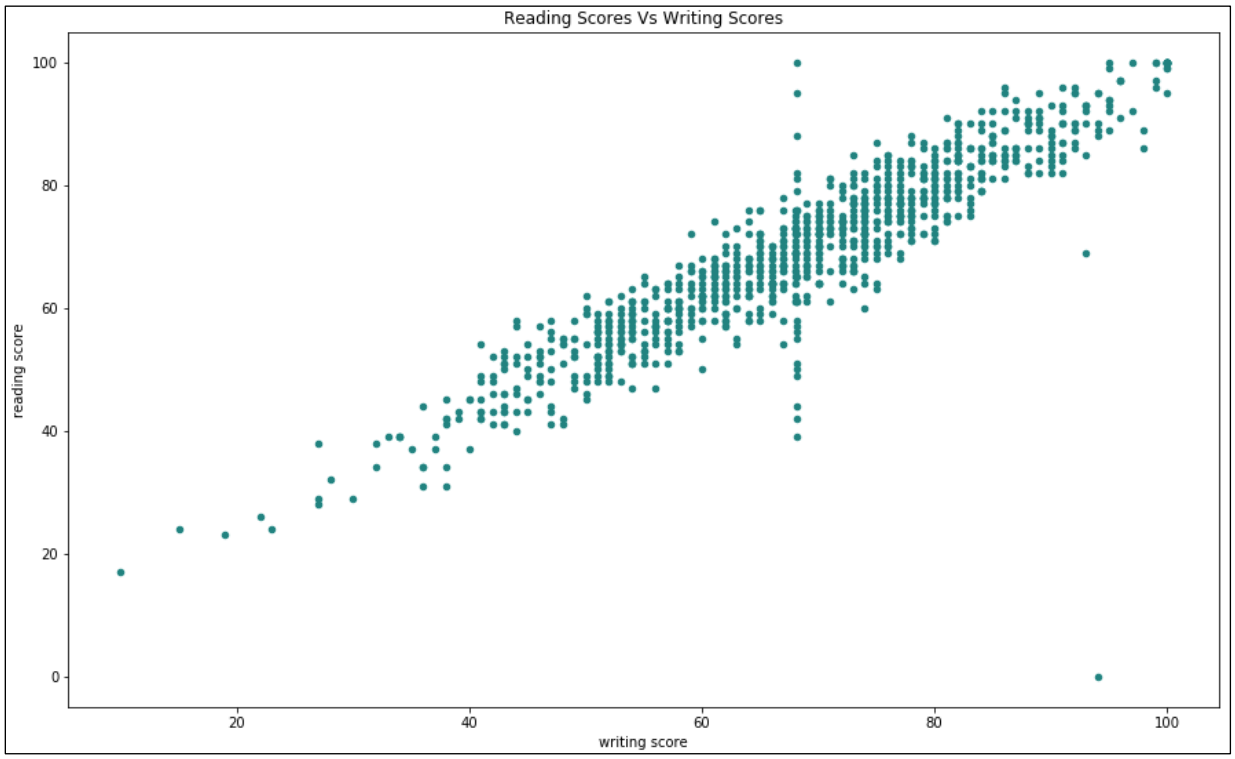
These outliers can be removed through an additional data manipulation stage if required.

Scatter Plot

To create a scatter plot within Python the **plot** method is used on the CAS table. The syntax includes an x-axis variable, y-axis variable and the type of plot which we would like to create. In addition to this, options have been added to choose a color for the plot and **figsize** to increase the size of the plot. Finally, a title can be added. The full code to complete this is as follows:

```
casstumod.plot(x='writing score', y='reading score', kind='scatter',
               color='teal', figsize=(15,9))
plt.title('Reading Scores Vs Writing Scores')
plt.show()
```

Output 20 shows the scatter plot which is created.



Output 20: Scatter plot created within Python.

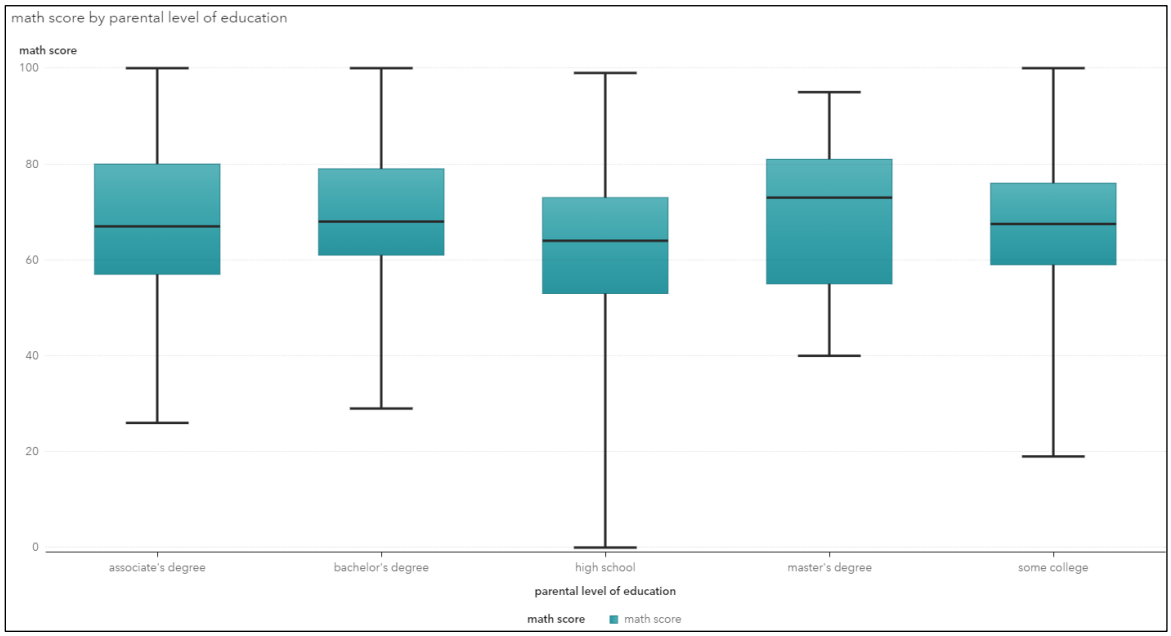
SAS VISUAL ANALYTICS

Another method of visualizing the data promoted to the public CAS library is to use SAS Visual Analytics. SAS Visual Analytics can produce the same results within a drag and drop interface that is simple to use without any SAS or even coding knowledge.

Box Plot

The box plot can be created using a drag and drop method with the variable math score and a grouping variable or parental level of education. The default coloring within SAS Visual Analytics are more visually appealing and display the exact same results as you would achieve within Python.

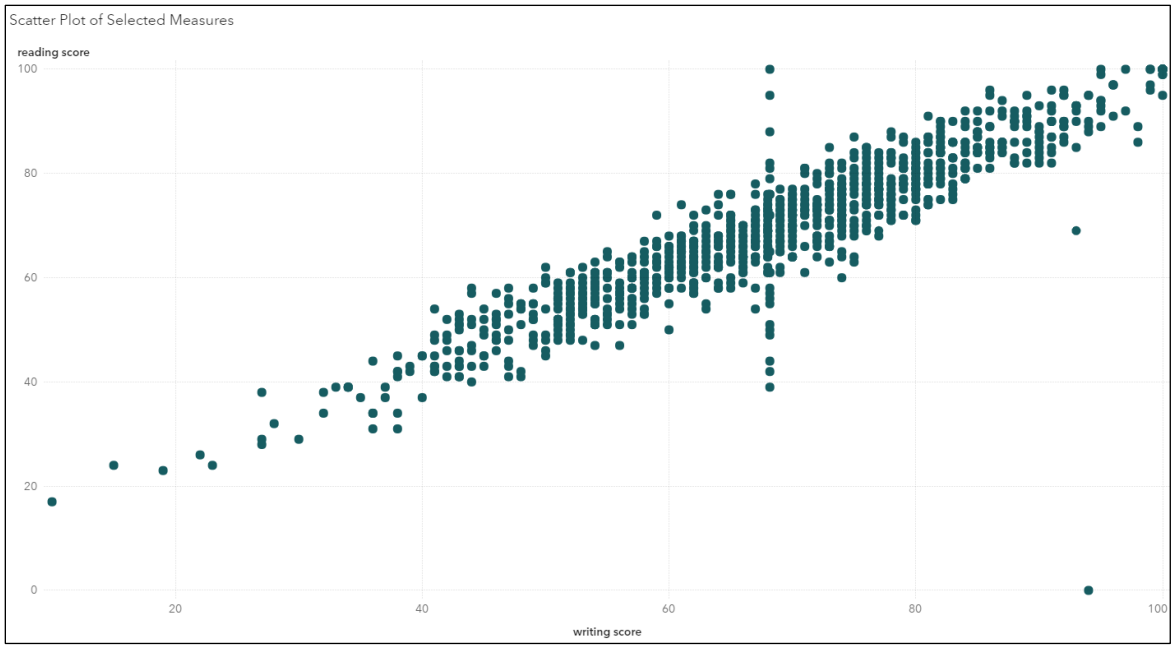
Output 21 displays the results achieved within SAS Visual Analytics.



Output 21: The default box plot created within SAS Visual Analytics.

Scatter Plot

The scatter plot can also be created through the drag and drop interface. This has been generated using a scatter plot object with the reading score and writing score as data objects. Output 22 displays the default results achieved.



Output 22: The default scatter plot created within SAS Visual Analytics.

THE POSSIBILITIES ARE ENDLESS...

SWAT allows Python users to truly gain the advantages of SAS Viya. They can now complete tasks in multiple interfaces allowing true collaboration across a unified analytics platform.

Table 4 displays an example of a list of tasks which can be completed in both interfaces. It is perfectly possible to jump between Python and SAS Viya web applications throughout the tasks listed. For example, the data can be read in with Python, some manipulation completed, and then further manipulation performed within SAS Viya Data Preparation. Later it can then be explored and visualized within the SAS Visual Analytics interface. All these tasks can be completed without any SAS code.

Example Tasks	Python	SAS Viya
Read in RAW data.	• SWAT provides CAS actions to read in data directly into a CAS library. • Python modules can read the data into a Python Data Frame that can be loaded into CAS later.	The SAS Data Explorer module can be used to drag and drop data to load into CAS.
Manipulate the data.	• SWAT provides CAS actions to implement on CAS tables. • Python libraries such as pandas are available to complete data analysis and manipulation tasks.	The SAS Data Preparation module can be used to modify the data.
Explore and Visualize data.	• SWAT provides CAS actions to create histograms and bar charts on the CAS table. • Python libraries such as matplotlib and seaborn can be used to create visualizations of a CAS table.	SAS Visual Analytics allows users to drag and drop objects to create custom reports.
Build a Model.	• SWAT provides CAS actions to create decision trees, regression and other models. • Python packages such as SciKit-Learn and TensorFlow are available to complete modelling.	SAS Model Studio can be used to create custom projects for Data Mining, Machine Learning, Forecasting and Text Analytics.

Table 4: Example tasks that can be completed in both SAS Viya applications and Python.

It is important to note, that not only can the tasks be completed in Python and SAS Viya applications, but SAS Studio can also be used to complete each of these tasks. SAS Studio Tasks and Utilities can allow some of this to be completed without any SAS code also.

CONCLUSION

SWAT encourages SAS and Python users to work in collaboration within the SAS Viya platform. Users can choose their preference in terms of programming language and interface which enables those who typically code within Python to continue to code in an interface that is familiar to them. The availability of SAS Visual Analytics enables users to gain access to the visualization and data science capabilities that the application has to offer and to harness the power and simplicity of the application without having to retrain in new software.

REFERENCES

ANACONDA. (2019) *Download Anaconda Distribution* [Online] Available from: https://www.anaconda.com/download/ [Accessed 21/03/19].

CALLAHAN, D. (2018) *Dan Callahan - Keynote - PyCon 2018*. [Online film] Available from: https://www.youtube.com/watch?v=ITksU31c1WY&feature=youtu.be&t=409 [Accessed 21/03/19]

FOREMAN, C. (2018) *SAS® and Python: The Perfect Partners in Crime* [Online] Available from: https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2597-2018.pdf [Accessed 21/03/19].

SAS (2016) *swat.CAS.upload_file* [Online] Available from: https://sassoftware.github.io/python-swat/generated/swat.cas.connection.CAS.upload_file.html [Accessed 21/03/19].

PYTHON. (2019) *Find, install and publish Python packages with the Python Package* Index [Online] Available from: https://pypi.org/ [Accessed 21/03/19].

RECOMMENDED READING

SAS (2017) *A Python interface to MVA SAS.* [Online] GitHub. Available from: https://github.com/sassoftware/saspy [Accessed 21/03/19].

SAS (2017) *SAS Kernel for Jupyter.* [Online] GitHub. Available from: https://github.com/sassoftware/sas_kernel [Accessed 21/03/19].

SAS (2017) *SAS Scripting Wrapper for Analytics Transfer (SWAT).* [Online] GitHub. Available from: https://github.com/sassoftware/python-swat/ [Accessed 21/03/19].

HEYNE, G (2017) *Using the Force of Python and SAS Viya on Star Wars Fan Posts*. [Online] Available from: https://analytics.ncsu.edu/sesug/2017/SESUG2017_Paper-170_Final_PDF.pdf [Accessed 21/03/19].

SMITH, K D and MENG, X (2017). *Using Python with SAS® Cloud Analytic Services (CAS)*. [Online] Available from: https://support.sas.com/resources/papers/proceedings17/SAS0152-2017.pdf [Accessed: Accessed 21/03/19].

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Carrie Foreman
Amadeus Software Limited
+44 (0) 1993848010
carrie.foreman@amadeus.co.uk
www.amadeus.co.uk