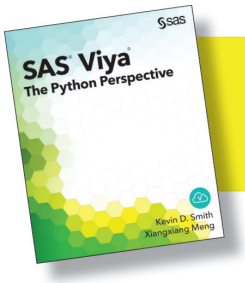
SAS® Viya®

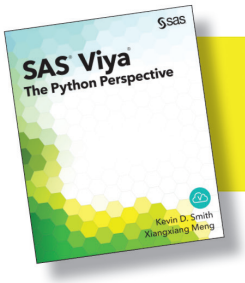
The Python Perspective

Kevin D. Smith
Xiangxiang Meng



From *SAS® Viya®: The Python Perspective.*
Full book available for purchase here.

Contents



From *SAS® Viya®: The Python Perspective.*
Full book available for purchase here.

Chapter 2: The Ten-Minute Guide to Using CAS from Python

If you are already familiar with Python, have a running CAS server, and just can't wait to get started, we've written this chapter just for you. This chapter is a very quick summary of what you can do with CAS from Python. We don't provide a lot of explanation of the examples; that comes in the later chapters. This chapter is here for those who want to dive in and work through the details in the rest of the book as needed.

In all of the sample code in this chapter, we are using the IPython interface to Python.

Importing SWAT and Getting Connected

The only thing you need to know about the CAS server in order to get connected is the host name, the port number, your user name, and your password. The SWAT package contains the CAS class that is used to communicate with the server. The arguments to the CAS class are hostname, port, username, and password[1], in that order. Note that you can use the REST interface by specifying the HTTP port that is used by the CAS server. The CAS class can autodetect the port type for the standard CAS port and HTTP. However, if you use HTTPS, you must specify protocol='https' as a keyword argument to the CAS constructor. You can also specify 'cas' or 'http' to explicitly override autodetection.

```
In [1]: import swat

In [2]: conn = swat.CAS('server-name.mycompany.com', 5570,
   ...:                 'username', 'password')
```

When you connect to CAS, it creates a session on the server. By default, all resources (CAS actions, data tables, options, and so on) are available only to that session. Some resources can be promoted to a global scope, which we discuss later in the book.

To see what CAS actions are available, use the help method on the CAS connection object, which calls the help action on the CAS server.

```
In [3]: out = conn.help()
NOTE: Available Action Sets and Actions:
NOTE:    accessControl
NOTE:       assumeRole - Assumes a role
NOTE:       dropRole - Relinquishes a role
NOTE:       showRolesIn - Shows the currently active role
NOTE:       showRolesAllowed - Shows the roles that a user
                               is a member of
NOTE:       isInRole - Shows whether a role is assumed
NOTE:       isAuthorized - Shows whether access is authorized
NOTE:       isAuthorizedActions - Shows whether access is
                                  authorized to actions
NOTE:       isAuthorizedTables - Shows whether access is authorized
                                 to tables
NOTE:       isAuthorizedColumns - Shows whether access is authorized
                                  to columns
NOTE:       listAllPrincipals - Lists all principals that have
                                explicit access controls
NOTE:       whatIsEffective - Lists effective access and
                              explanations (Origins)


NOTE:       partition - Partitions a table
NOTE:       recordCount - Shows the number of rows in a Cloud
                          Analytic Services table
NOTE:       loadDataSource - Loads one or more data source interfaces
NOTE:       update - Updates rows in a table
```

The printed notes describe all of the CAS action sets and the actions in those action sets. The help action also returns the action set and action information as a return value. The return values from all actions are in the form of CASResults objects, which are a subclass of the Python collections.OrderedDict class. To see a list of all of the keys, use the keys method just as you would with any Python dictionary. In this case, the keys correspond to the names of the CAS action sets.

```
In [4]: list(out.keys())
Out[4]:
['accessControl',
 'builtins',
 'configuration',
 'dataPreprocess',
 'dataStep',
 'percentile',
 'search',
 'session',
 'sessionProp',
 'simple',
 'table']
```

Printing the contents of the return value shows all of the top-level keys as sections. In the case of the help action, the information about each action set is returned in a table in each section. These tables are stored in the dictionary as Pandas DataFrames.

```
In [5]: out
Out[5]:
[accessControl]
```

```
                    name                                description
 0            assumeRole                             Assumes a role
 1              dropRole                         Relinquishes a role
 2          showRolesIn           Shows the currently active role
 3      showRolesAllowed  Shows the roles that a user is a mem...
 4              isInRole           Shows whether a role is assumed
 5         isAuthorized        Shows whether access is authorized
 6   isAuthorizedActions  Shows whether access is authorized t...
 7    isAuthorizedTables  Shows whether access is authorized t...
 8   isAuthorizedColumns  Shows whether access is authorized t...
 9     listAllPrincipals  Lists all principals that have expli...
 10      whatIsEffective  Lists effective access and explanati...
 11          listAcsData  Lists access controls for caslibs, t...
 12     listAcsActionSet  Lists access controls for an action ...
 13      repAllAcsCaslib  Replaces all access controls for a c...
 14       repAllAcsTable  Replaces all access controls for a t...
 15      repAllAcsColumn  Replaces all access controls for a c...
 16   repAllAcsActionSet  Replaces all access controls for an ...
 17      repAllAcsAction  Replaces all access controls for an ...
 18     updSomeAcsCaslib  Adds, deletes, and modifies some acc...
 19      updSomeAcsTable  Adds, deletes, and modifies some acc...

 ... truncated ...

+ Elapsed: 0.0034s, user: 0.003s, mem: 0.164mb
```

Since the output is based on Python's dictionary object, you can access each key individually as well.

```
In [6]: out['builtins']
Out[6]:
                name                                description
0            addNode                Adds a machine to the server
1         removeNode  Remove one or more machines from the...
2               help  Shows the parameters for an action o...
3          listNodes  Shows the host names used by the server
4      loadActionSet  Loads an action set for use in this ...
5   installActionSet  Loads an action set in new sessions ...
6                log          Shows and modifies logging levels
7     queryActionSet    Shows whether an action set is loaded
8          queryName  Checks whether a name is an action o...
9            reflect  Shows detailed parameter information...
10      serverStatus            Shows the status of the server
11             about            Shows the status of the server
12          shutdown                      Shuts down the server
13          userInfo  Shows the user information for your ...
14     actionSetInfo  Shows the build information from loa...
15           history  Shows the actions that were run in t...
16         casCommon  Provides parameters that are common ...
17              ping  Sends a single request to the server...
18              echo  Prints the supplied parameters to th...
19       modifyQueue  Modifies the action response queue s...
20    getLicenseInfo  Shows the license information for a ...
21    refreshLicense  Refresh SAS license information from...
22       httpAddress  Shows the HTTP address for the serve...
```

The keys are commonly alphanumeric, so the CASResults object was extended to enable you to access keys as attributes as well. This just keeps your code a bit cleaner. However, you should be aware that if a result key has the same name as a Python dictionary method, the dictionary method takes precedence. In the following code, we access the builtins key again, but this time we access it as if it were an attribute.

```
In [7]: out.builtins
Out[7]:
                 name                                 description
0             addNode                 Adds a machine to the server
1          removeNode  Remove one or more machines from the...
2                help  Shows the parameters for an action o...
3           listNodes  Shows the host names used by the server
4       loadActionSet  Loads an action set for use in this ...
5    installActionSet  Loads an action set in new sessions ...
6                 log         Shows and modifies logging levels
7      queryActionSet    Shows whether an action set is loaded
8           queryName  Checks whether a name is an action o...
9             reflect  Shows detailed parameter information...
10      serverStatus            Shows the status of the server
11             about            Shows the status of the server
12          shutdown                      Shuts down the server
13          userInfo  Shows the user information for your ...
14     actionSetInfo  Shows the build information from loa...
15           history  Shows the actions that were run in t...
16         casCommon  Provides parameters that are common ...
17              ping  Sends a single request to the server...
18              echo  Prints the supplied parameters to th...
19       modifyQueue  Modifies the action response queue s...
20    getLicenseInfo  Shows the license information for a ...
21    refreshLicense  Refresh SAS license information from...
22       httpAddress  Shows the HTTP address for the serve...
```

Running CAS Actions

Just like the help action, all of the action sets and actions are available as attributes and methods on the CAS connection object. For example, the userinfo action is called as follows.

```
In [8]: conn.userinfo()
Out[8]:
[userInfo]

 {'anonymous': False,
  'groups': ['users'],
  'hostAccount': True,
  'providedName': 'username',
  'providerName': 'Active Directory',
  'uniqueId': 'username',
  'userId': 'username'}

+ Elapsed: 0.000291s, mem: 0.0826mb
```

The result this time is a CASResults object, the contents of which is a dictionary under a single key (userInfo) that contains information about your user account. Although all actions return a CASResults object, there are no strict rules about what keys and values are in that object. The returned values are

determined by the action and vary depending on the type of information returned. Analytic actions typically return one or more DataFrames. If you aren't using IPython to format your results automatically, you can cast the result to a dictionary and then print it using pprint for a nicer representation.

```
In [9]: from pprint import pprint
In [10]: pprint(dict(conn.userinfo()))
{'userInfo': {'anonymous': False,
              'groups': ['users'],
              'hostAccount': True,
              'providedName': 'username',
              'providerName': 'Active Directory',
              'uniqueId': 'username',
              'userId': 'username'}}
```

When calling the help and userinfo actions, we actually used a shortcut. In some cases, you might need to specify the fully qualified name of the action, which includes the action set name. This can happen if two action sets have an action of the same name, or an action name collides with an existing method or attribute name on the CAS object. The userinfo action is contained in the builtins action set. To call it using the fully qualified name, you use builtins.userinfo rather than userinfo on the CAS object. The builtins level in this call corresponds to a CASActionSet object that contains all of the actions in the builtins action set.

```
In [11]: conn.builtins.userinfo()
```

The preceding code provides you with the same result as the previous example does.

Loading Data

The easiest way to load data into a CAS server is by using the upload method on the CAS connection object. This method uses a file path or URL that points to a file in various possible formats including CSV, Excel, and SAS data sets. You can also pass a Pandas DataFrame object to the upload method in order to upload the data from that DataFrame to a CAS table. We use the classic Iris data set in the following data loading example.

```
In [12]: out = conn.upload('https://raw.githubusercontent.com/' +
   ....:                   'pydata/pandas/master/pandas/tests/' +
   ....:                   'data/iris.csv')

In [13]: out
Out[13]:
[caslib]

 'CASUSER(username)'

[tableName]

 'IRIS'

[casTable]

 CASTable('IRIS', caslib='CASUSER(username)')

+ Elapsed: 0.0629s, user: 0.037s, sys: 0.021s, mem: 48.4mb
```

The output from the upload method is, again, a CASResults object. The output contains the name of the created table, the CASLib that the table was created in, and a CASTable object that can be used to interact with the table on the server. CASTable objects have all of the same CAS action set and action methods of the connection that created it. They also include many of the methods that are defined by Pandas DataFrames so that you can operate on them as if they were local DataFrames. However, until you explicitly fetch the data or call a method that returns data from the table (such as head or tail), all operations are simply combined on the client side (essentially creating a client-side view) until they are needed for the call to the CAS server for data.

We can use actions such as tableinfo and columninfo to access general information about the table itself and its columns.

```
# Store CASTable object in its own variable.
In [14]: iris = out.casTable

# Call the tableinfo action on the CASTable object.
In [15]: iris.tableinfo()
Out[15]:
[TableInfo]

    Name  Rows  Columns Encoding CreateTimeFormatted  \
 0  IRIS   150        5    utf-8  01Nov2016:16:38:59

      ModTimeFormatted JavaCharSet    CreateTime       ModTime  \
 0  01Nov2016:16:38:59        UTF8  1.793638e+09  1.793638e+09

    Global  Repeated  View SourceName SourceCaslib  Compressed  \
 0       0         0     0                                   0

   Creator Modifier
 0  username

+ Elapsed: 0.000856s, mem: 0.104mb

# Call the columninfo action on the CASTable.
In [16]: iris.columninfo()
Out[16]:
[ColumnInfo]

         Column  ID     Type  RawLength  FormattedLength  NFL  NFD
 0  SepalLength   1   double          8               12    0    0
 1   SepalWidth   2   double          8               12    0    0
 2  PetalLength   3   double          8               12    0    0
 3   PetalWidth   4   double          8               12    0    0
 4         Name   5  varchar         15               15    0    0

+ Elapsed: 0.000727s, mem: 0.175mb
```

Now that we have some data, let's run some more interesting CAS actions on it.

Executing Actions on CAS Tables

The simple action set that comes with CAS contains some basic analytic actions. You can use either the help action or the IPython **?** operator to view the available actions.

```
In [17]: conn.simple?
Type:        Simple
String form: <swat.cas.actions.Simple object at 0x4582b10>
File: swat/cas/actions.py
Definition:  conn.simple(self, *args, **kwargs)
Docstring:
Analytics

Actions
-------
simple.correlation : Generates a matrix of Pearson product-moment
                     correlation coefficients
simple.crosstab    : Performs one-way or two-way tabulations
simple.distinct    : Computes the distinct number of values of the
                     variables in the variable list
simple.freq        : Generates a frequency distribution for one or
                     more variables
simple.groupby     : Builds BY groups in terms of the variable value
                     combinations given the variables in the variable
                     list
simple.mdsummary   : Calculates multidimensional summaries of numeric
                     variables
simple.numrows     : Shows the number of rows in a Cloud Analytic
                     Services table
simple.paracoord   : Generates a parallel coordinates plot of the
                     variables in the variable list
simple.regression  : Performs a linear regression up to 3rd-order
                     polynomials
simple.summary     : Generates descriptive statistics of numeric
                     variables such as the sample mean, sample
                     variance, sample size, sum of squares, and so on
simple.topk        : Returns the top-K and bottom-K distinct values of
                     each variable included in the variable list based
                     on a user-specified ranking order
```

Let's run the summary action on our CAS table.

```
In [18]: summ = iris.summary()

In [19]: summ
Out[19]:
[Summary]

 Descriptive Statistics for IRIS

         Column  Min  Max      N  NMiss      Mean    Sum       Std  \
 0  SepalLength  4.3  7.9  150.0    0.0  5.843333  876.5  0.828066
 1   SepalWidth  2.0  4.4  150.0    0.0  3.054000  458.1  0.433594
 2  PetalLength  1.0  6.9  150.0    0.0  3.758667  563.8  1.764420
 3   PetalWidth  0.1  2.5  150.0    0.0  1.198667  179.8  0.763161
```

```
      StdErr       Var      USS         CSS         CV     TValue  \
0  0.067611  0.685694  5223.85  102.168333  14.171126  86.425375
1  0.035403  0.188004  1427.05   28.012600  14.197587  86.264297
2  0.144064  3.113179  2583.00  463.863733  46.942721  26.090198
3  0.062312  0.582414   302.30   86.779733  63.667470  19.236588

            ProbT
0  3.331256e-129
1  4.374977e-129
2   1.994305e-57
3   3.209704e-42

+ Elapsed: 0.0256s, user: 0.019s, sys: 0.009s, mem: 1.74mb
```

The summary action displays summary statistics in a form that is familiar to SAS users. If you want them in a form similar to what Pandas users are used to, you can use the describe method (just like on DataFrames).

```
In [20]: iris.describe()
Out[20]:
       SepalLength  SepalWidth  PetalLength  PetalWidth
count   150.000000  150.000000   150.000000  150.000000
mean      5.843333    3.054000     3.758667    1.198667
std       0.828066    0.433594     1.764420    0.763161
min       4.300000    2.000000     1.000000    0.100000
25%       5.100000    2.800000     1.600000    0.300000
50%       5.800000    3.000000     4.350000    1.300000
75%       6.400000    3.300000     5.100000    1.800000
max       7.900000    4.400000     6.900000    2.500000
```

Note that when you call the describe method on a CASTable object, it calls various CAS actions in the background to do the calculations. This includes the summary, percentile, and topk actions. The output of those actions is combined into a DataFrame in the same form that the real Pandas DataFrame describe method returns. This enables you to use CASTable objects and DataFrame objects interchangeably in your workflow for this method and many other methods.

Data Visualization

Since the tables that come back from the CAS server are subclasses of Pandas DataFrames, you can do anything to them that works on DataFrames. You can plot the results of your actions using the plot method or use them as input to more advanced packages such as Matplotlib and Bokeh, which are covered in more detail in a later section.

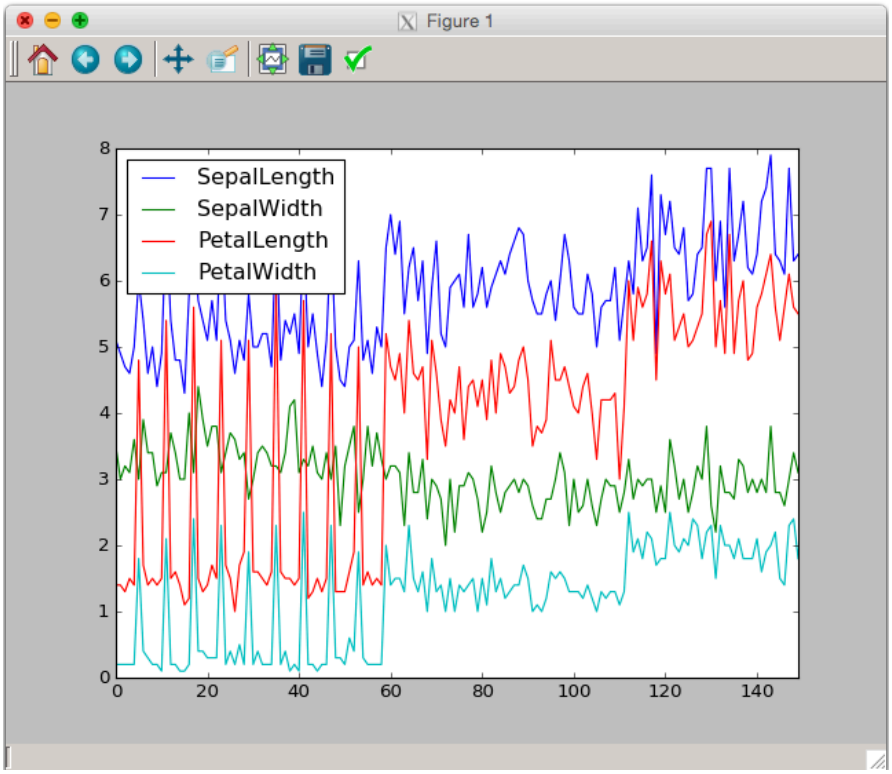
The following example uses the plot method to download the entire data set and plot it using the default options.

```
In [21]: iris.plot()
Out[21]: <matplotlib.axes.AxesSubplot at 0x5339050>
```

If the plot doesn't show up automatically, you might have to tell Matplotlib to display it.

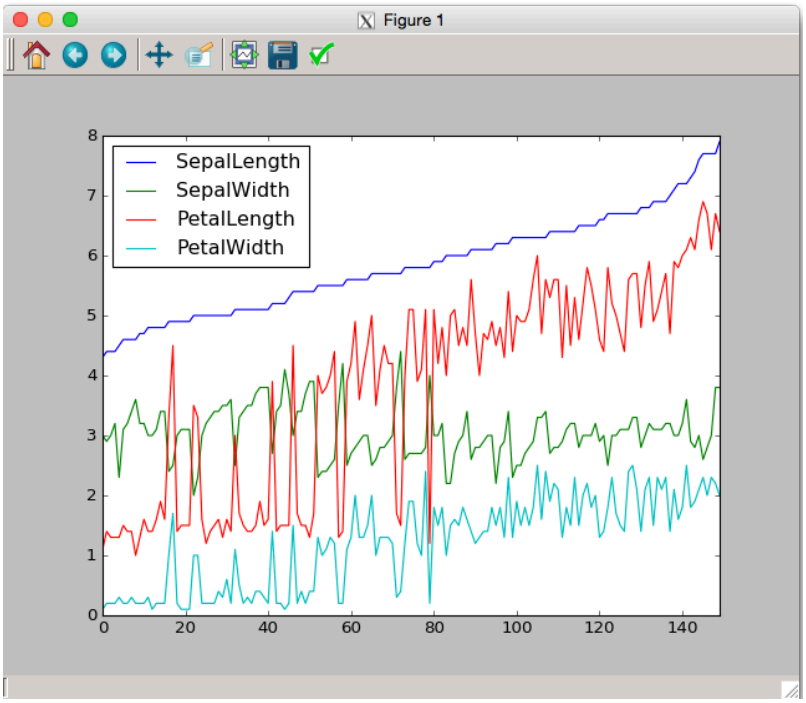
```
In [22]: import matplotlib.pyplot as plt

In [23]: plt.show()
```

The output that is created by the plot method follows.



Even if you loaded the same data set that we have used in this example, your plot might look different since CAS stores data in a distributed manner. Because of this, the ordering of data from the server is not guaranteed unless you sort it when it is fetched. If you run the following commands, you plot the data sorted by SepalLength and SepalWidth.

```
In [24]: iris.sort_values(['SepalLength', 'SepalWidth']).plot()
```



Closing the Connection

As with any network or file resource in Python, you should close your CAS connections when you are finished. They time out and disappear eventually if left open, but it's always a good idea to clean them up explicitly.

```
In [25]: conn.close()
```

Conclusion

Hopefully this 10-minute guide was enough to give you an idea of the basic workflow and capabilities of the Python CAS client. In the following chapters, we dig deeper into the details of the Python CAS client and how to blend the power of SAS analytics with the tools that are available in the Python environment.

---

[1] Later in the book, we show you how to store your password so that you do not need to specify it in your programs.

About This Book

What Does This Book Cover?

This book is an introduction to using the Python client on the SAS Viya platform. SAS Viya is a high-performance, fault-tolerant analytics architecture that can be deployed on both public and private cloud infrastructures. Although SAS Viya can be used by various SAS applications, it also enables you to access analytic methods from SAS, Python, Lua, and Java, as well as through a REST interface using HTTP or HTTPS. Of course, in this book we focus on the perspective of SAS Viya from Python.

SAS Viya consists of multiple components. The central piece of this ecosystem is SAS Cloud Analytic Services (CAS). CAS is the cloud-based server that all clients communicate with to run analytical methods. The Python client is used to drive the CAS component directly using objects and constructs that are familiar to Python programmers.

We assume that you have some knowledge about Python before you approach the topics in this book. However, the book includes an appendix that covers the features of Python that are used in the CAS Python client. We do not assume any knowledge of CAS itself. However, you must have a CAS server that is set up and is running in order to execute the examples in this book.

The chapters in the first part of the book cover topics from installation of Python to the basics of connecting, loading data, and getting simple analyses from CAS. Depending on your familiarity with Python, after reading the "Ten-Minute Guide to Using CAS from Python," you might feel comfortable enough to jump to the chapters later in the book that are dedicated to statistical methods. However, the chapters in the middle of the book cover more detailed information about working with CAS such as constructing action calls to CAS and processing the results, error handling, managing your data in CAS, and using object interfaces to CAS actions and CAS data tables. Finally, the last chapter about advanced topics covers features and workflows that you might want to take advantage of when you are more experienced with the Python client.

This book covers topics that are useful to complete beginners as well as to experienced CAS users. Its examples extend from creating connections to CAS to simple statistics and machine learning. The book is also useful as a desktop reference.

Is This Book for You?

If you are using the SAS Viya platform in your work and you want to access analytics from SAS Cloud Analytic Services (CAS) using Python, then this book is a great starting point. You'll learn about general CAS workflows, as well as the Python client that is used to communicate with CAS.

What Are the Prerequisites for This Book?

Some Python experience is definitely helpful while reading this book. If you do not know Python, there is an appendix that gives a crash course in learning Python. There are also a multitude of resources on the Internet for learning Python. The later chapters in the book cover data analysis and modeling topics. Although the examples provide step-by-step code walk-throughs, some training about these topics beforehand is helpful.

Scope of This Book

This book covers the installation and usage of the Python client for use with CAS. It does not cover the installation, configuration, and maintenance of CAS itself.

What Should You Know about the Examples?

This book includes tutorials for you to follow to gain "hands-on" experience with SAS.

Software Used to Develop the Book's Content

This book was written using version 1.0.0 of the SAS Scripting Wrapper for Analytics Transfer (SWAT) package for Python. SAS Viya 3.1 was used. Various Python resources and packages were used as well. SWAT works with many versions of these packages. The URLs of SWAT and other resources are shown as follows:

SAS Viya
`www.sas.com/en_us/software/viya.html`

SAS Scripting Wrapper for Analytics Transfer (SWAT) – Python client to CAS
`github.com/sassoftware/python-swat` (GitHub repository)
`sassoftware.github.io/python-swat/` (documentation)

Python
`www.python.org/`

Anaconda – Data Science Python Distribution by Continuum Analytics
`www.continuum.io/`

Pandas – Python Data Analysis Library
`pandas.pydata.org/`

Jupyter – Scientific notebook application
`jupyter.org/`

Example Code and Data

You can access the example code and data for this book by linking to its author page at https://support.sas.com/authors or on GitHub at: https://github.com/sassoftware/sas-viya-the-python-perspective.

We Want to Hear from You

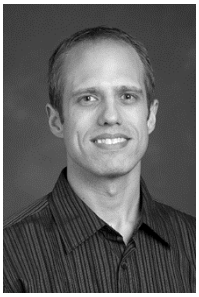
SAS Press books are written *by* SAS Users *for* SAS Users. We welcome your participation in their development and your feedback on SAS Press books that you are using. Please visit https://www.sas.com/books to do the following:

- Sign up to review a book
- Recommend a topic
- Request information about how to become a SAS Press author
- Provide feedback on a book

Do you have questions about a SAS Press book that you are reading? Contact the author through saspress@sas.com or https://support.sas.com/author_feedback.

SAS has many resources to help you find answers and expand your knowledge. If you need additional help, see our list of resources: https://www.sas.com/books.

About These Authors



Kevin D. Smith has been a software developer at SAS since 1997. He has been involved in the development of PROC TEMPLATE and other underlying ODS technologies for most of his tenure. He has spoken at numerous SAS Global Forum conferences, as well as at regional and local SAS users groups with the “From Scratch” series of presentations that were created to help users of any level master various ODS technologies. More recently, he has been involved in the creation of the scripting language interfaces to SAS Cloud Analytic Services on the SAS Viya platform.

Xiangxiang Meng, PhD, is a Senior Product Manager at SAS. The current focus of his work is on SAS® Visual Statistics, cognitive computing, the Python interface to SAS Cloud Analytic Services, and other new product initiatives. Previously, Xiangxiang worked on SAS® LASR™ Analytic Server, SAS® In-Memory Statistics for Hadoop, SAS Recommendation Systems, and SAS® Enterprise Miner™. His research interests include decision trees and tree ensemble models, automated and cognitive pipelines for business intelligence and machine learning, and parallelization of machine learning algorithms on distributed data. Xiangxiang received his PhD and MS from the University of Cincinnati.

Learn more about these authors by visiting their author pages, where you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more:
http://support.sas.com/smithk
http://support.sas.com/meng