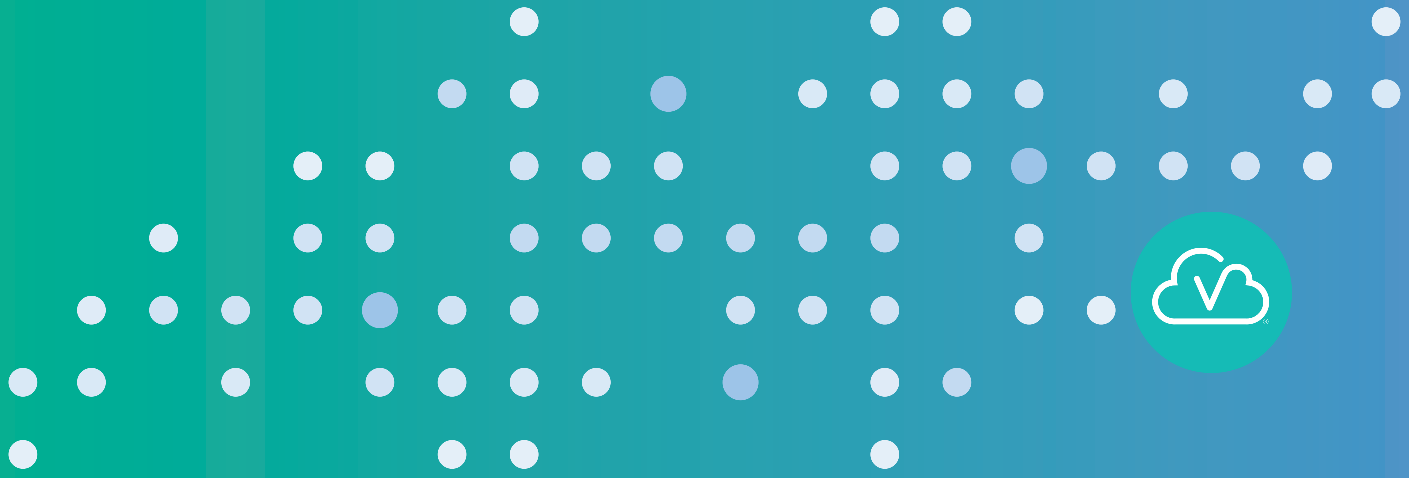
Exploring SAS® Viya®

Programming and Data Management



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *Exploring SAS® Viya®: Programming and Data Management*. Cary, NC: SAS Institute Inc.

Exploring SAS® Viya®: Programming and Data Management

978-1-64295-483-8 (Paperback)
978-1-64295-336-7 (Web PDF)

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

April 2019

Contents

About This Book

What Does This Book Cover?

SAS® Viya® is an open analytics platform that can handle any data type, volume, or speed. A cloud-enabled, in-memory analytics engine, it is elastic, scalable and fault tolerant. It contains a standardized code base that supports programming in SAS and other languages, such as Python, R, Java and Lua. In addition, it can deploy seamlessly to any infrastructure or application ecosystem with support for cloud, on-site or hybrid environment. The high-performance processing power of SAS Viya is provided by SAS Cloud Analytics Services, or CAS. CAS is an in-memory engine that can dramatically accelerate data management and analytics with SAS. There are many products offered by SAS that are powered by SAS Viya including:

- SAS Visual Data Mining and Machine Learning
- SAS Data Preparation
- SAS Visual Analytics
- SAS Visual Statistics
- And more!

SAS Viya is designed to coexist with SAS 9.4 solutions and the SAS 9 environment. While SAS 9 and SAS Viya are two run-time environments built for different use cases, you can make your SAS 9.4 data available to SAS Viya. These environments also share some functionality. For example, SAS 9 uses the SAS programming language, and SAS Viya uses the next generation of SAS programming with the new CAS programming language. The CAS language is very similar to the SAS language. Some procedures are available in both SAS 9 and SAS Viya, so some existing SAS code can be run in SAS Viya. However, SAS Viya also contains new procedures that take advantage of the open, distributed environment. As a result, some SAS 9 procedures do not exist in SAS Viya.

It is easy to connect to SAS Viya's CAS to submit code. To write and run SAS code through your web browser, you can use the SAS Studio interface. With SAS Studio, you can access your data files, libraries, and existing programs and write new programs. SAS Viya uses PROC CAS to run CAS actions in SAS Cloud Analytic Services. You can use the REST APIs for any client language to access SAS analytics, data, and services. You can also use programming interfaces for Python, Java, and Lua to access this CAS functionality. In addition, you can continue to submit SAS code in batch mode.

The content in this book is based on SAS® Viya® Enablement, a free course available from SAS Education. This book covers how to access data files, libraries, and existing code in SAS Studio. You will also learn about new procedures in SAS Viya, how to write new code, as well as how to use some of the pre-installed tasks that come with SAS Visual Data Mining and Machine Learning. In the last chapter, you will learn how to use the features in SAS Data Preparation to perform data management tasks using SAS Data Explorer, SAS Data Studio, and SAS Lineage Viewer.

Is This Book for You?

If you are a SAS programmer transitioning from SAS 9 to SAS Viya, then this book is for you. You can use all of your existing SAS programming expertise in this new, high-powered SAS environment.

SAS Viya extends the SAS Platform to enable everyone – data scientists, business analysts, developers and executives alike – to collaborate and realize innovative results faster. If you are curious about SAS Viya and want to learn more about some of its features and capabilities, then this book is also for you.

What Should You Know about the Examples?

The content in this book is based on SAS® Viya® Enablement, a free course available from SAS Education. You can follow along with the examples in real time by watching the videos if you prefer.

This book includes tutorials for you to follow to gain hands-on experience with SAS Viya and SAS 9.4M5. Wherever possible, the source of the sample data is provided in a link. Some features shown may only be available if your site has licensed that feature in SAS Viya. Therefore, the options in your version of SAS may look different.

We Want to Hear from You

Chapter 1: SAS Viya Deployment

Introduction

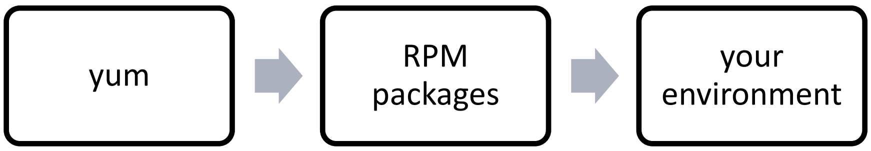
The high-performance processing power of the SAS Viya platform is provided by SAS Cloud Analytics Services (CAS). CAS is an in-memory engine that can dramatically accelerate data management and analytics with SAS. Some of the benefits of CAS include:

- CAS can run on a single machine or as a distributed server on multiple machines.
- Servers are multi-threaded, which means that data can be distributed to multiple CPU cores, with each core assigned a subset of the rows. All cores then process their designated rows at the same time, which is known as parallel processing.
- The distributed server has a communication layer that supports fault tolerance. This means that even if connectivity is lost to one or more threads, the server can still continue processing by distributing work to other functioning threads.
- The CAS server loads and processes data in-memory, which contributes to the blazing speed of SAS Viya. Data can come from SAS data sets, server-side files, event stream processing, and database files.
- The CAS server can also manage all of your data and easily share data with multiple users.
- CAS is scalable, which means it is elastic, allowing your cloud environment to expand or contract as processing needs change.

In this chapter, we look at the tools and utilities for deploying SAS Viya products, several possible topologies, and some deployment options with Hadoop.

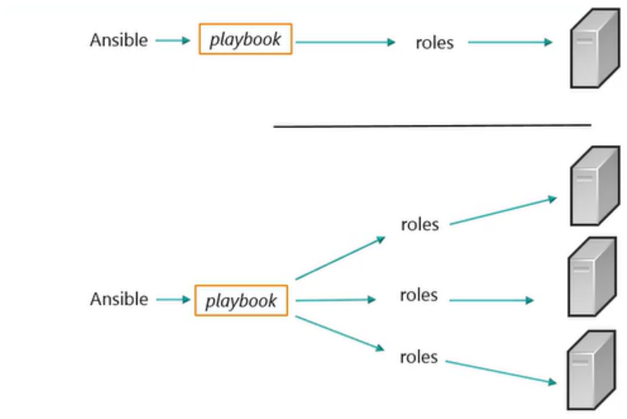
Deployment

Deployment of SAS Viya uses industry-standard deployment software such as Ansible and yum. SAS provides software as RPM packages, and uses the Linux utility yum to install the RPM packages in your environment.



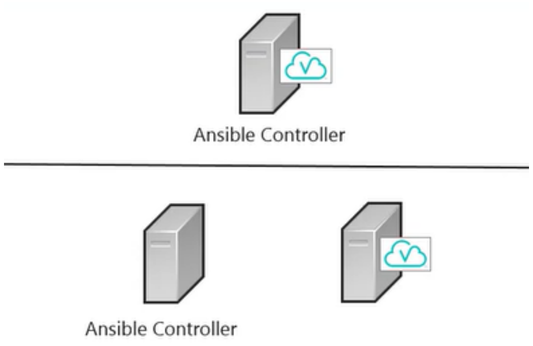
Ansible automates a series of yum commands to install the RPM packages on the machines that you designate. It uses a configuration management script called a playbook that maps a machine (or groups of machines) to well-defined roles, which associate groups of services to specific machines. To support Ansible, SAS provides a utility to generate a playbook that you customize for your environment, as shown in Figure 1.1.

Figure 1.1: Playbook Utility



The machine where Ansible is installed is called the *Ansible controller machine*. Ansible might be on the same machine as SAS Viya, or on a separate machine. (See Figure 1.2). This can simplify a multi-machine deployment because Ansible only needs to be installed on a single machine (the Ansible controller machine).

Figure 1.2: Ansible Controller Machine Configurations



Ansible can use SSH to deliver instructions and retrieve results from the other machines in the installation, called *managed nodes*.

Topologies

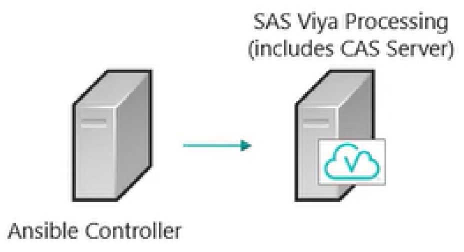
There are several possible topologies that you can use to deploy SAS Viya. Let's look at some examples.

Single Machine Deployment

In this first scenario, you are deploying all SAS software to a single machine. There are two options for such a deployment. You can separate the Ansible controller from the target node, or you can run Ansible on the same machine as the target.

SAS Cloud Analytics Services (CAS) includes the in-memory run-time server for SAS Viya products. You can potentially improve the performance of analytical processing by deploying CAS to its own machine or machines. In a simple scenario, a single machine handles all CAS operations, which include in-memory run-time analytics and supporting services. This is symmetric multiprocessing (SMP) architecture as shown in Figure 1.3.

Figure 1.3: SMP Architecture



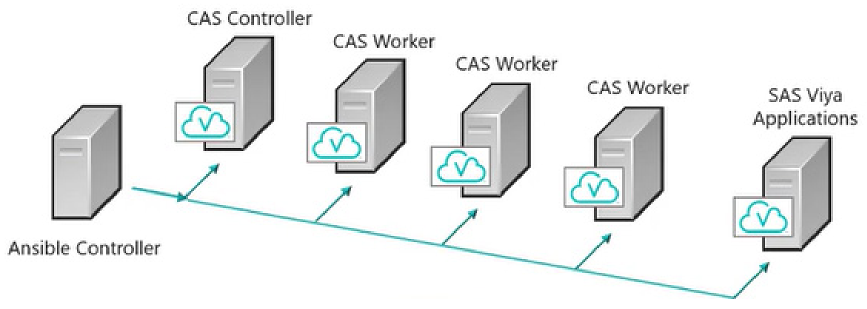
Multiple Machine Deployment

In the next scenario, a separate Ansible controller machine deploys the same SAS Viya software to multiple target machines. This is a good option for development, testing, staging, and production environments, or setting up the same deployment for different groups of users.

CAS can also be configured in distributed mode. This is massively parallel processing (MPP) architecture, which is a core design feature of SAS Viya. This scenario provides optimal processing capabilities. The CAS Controller distributes work to each of the CAS worker nodes, and the worker nodes send the results of the computations back to the CAS controller. (See Figure 1.4).

In this configuration, the node labeled SAS Viya Applications provides infrastructure support for SAS products, such as reporting and administrative services for web applications.

Figure 1.4: MPP Architecture



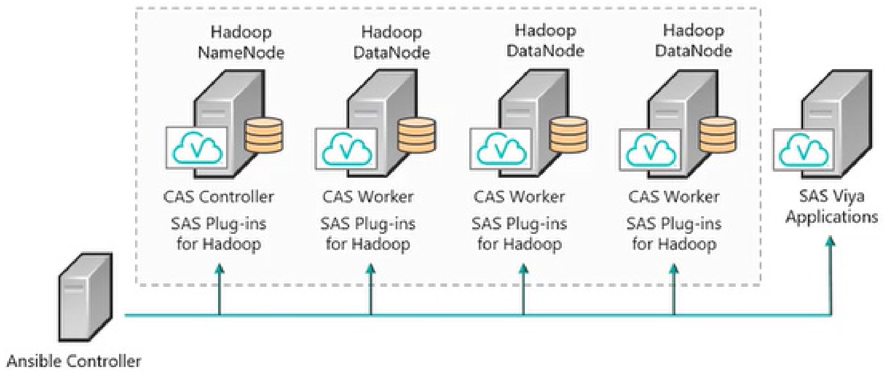
Hadoop

In both SMP and MPP architecture, the CAS server is multi-threaded for high performance. CAS servers are optimized to work jointly with Hadoop. You can connect to a Hadoop cluster in two ways: a co-located deployment and a remote deployment.

Co-located Deployment

Co-located deployments install the CAS in-memory run-time server onto an existing Hadoop cluster. The CAS controller software is installed on the Hadoop NameNode, and the CAS worker software is installed on the Hadoop DataNodes as shown in Figure 1.5. Notice that other SAS Viya applications are deployed to a separate machine.

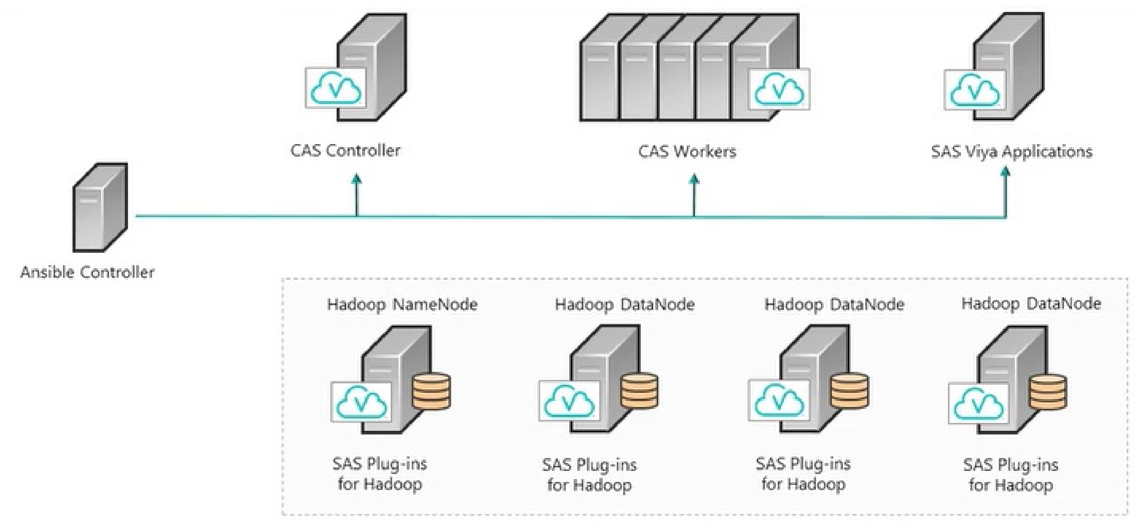
Figure 1.5: Co-located Deployment


Hadoop NameNode
Hadoop DataNode
Hadoop DataNode
Hadoop DataNode
CAS Controller SAS Plug-ins for Hadoop
CAS Worker SAS Plug-ins for Hadoop
CAS Worker SAS Plug-ins for Hadoop
CAS Worker SAS Plug-ins for Hadoop
SAS Viya Applications
Ansible Controller

Remote Deployment

Remote deployments pair CAS controller nodes and worker nodes on one set of machines with name and data nodes on a remote Hadoop cluster. Similar to the co-located deployment configuration, SAS Viya applications are deployed to a separate machine as shown in Figure 1.6.

Figure 1.6: Hadoop Deployment



SAS Viya Embedded Processes can run on Hadoop or Teradata machines to provide a computational engine near the data. This reduces unnecessary data movement and speeds up model scoring. SAS plug-ins for Hadoop provide connection and configuration information and can vary based on the Hadoop distribution that is used.

Connecting to the CAS Server in SAS 9 Clients

If you currently have the latest maintenance version of SAS 9, you can take advantage of direct interoperability between your clients and SAS Viya. All SAS 9 programming clients can submit code directly to the SAS Viya engine for optimized analytic processing including:

- SAS Studio
- SAS Enterprise Guide
- SAS Windowing Environment
- SAS Data Integration Studio

Even if you are running an earlier version of SAS, you can still use SAS/CONNECT technology to remotely execute code and transfer data to and from SAS Viya.

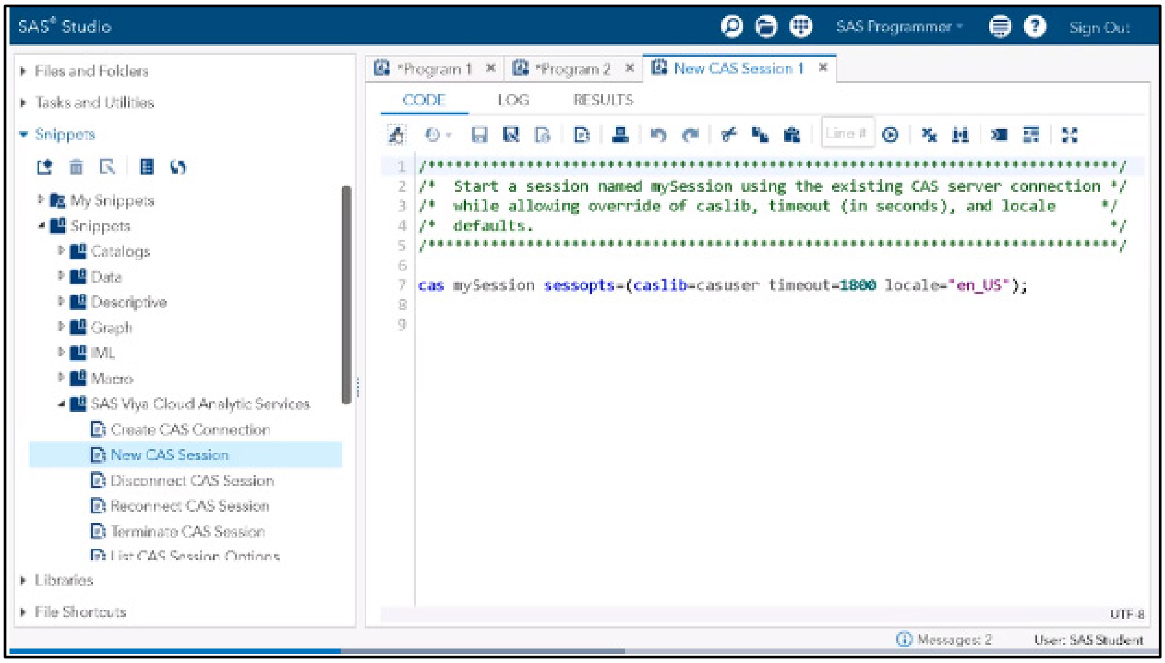
Let's look at how easy it is to connect to SAS Viya's Cloud Analytics Services to submit code that will load and process data in-memory.

SAS Studio Example

We will look at an example using SAS Studio, but the code will be exactly the same in SAS Enterprise Guide 7.15 or SAS 9.4M5.

Open your SAS Studio session. Remember, SAS 9 is the default server. If you want to take advantage of SAS Viya and the CAS server, you can start a new CAS Session. To open the new CAS session, you can open the New CAS Session snippet under Snippets – SAS Viya Cloud Analytics Services and double-click to send the code to the code window without any changes. See Figure 1.7.

Figure 1.7: Start New CAS Session



```
/****************************************************************************/
/*  Start a session named mySession using the existing CAS server connection */
/*  while allowing override of caslib, timeout (in seconds), and locale      */
/*  defaults.                                                                 */
/****************************************************************************/

cas mySession sessopts=(caslib=casuser timeout=1800 locale="en_US");
```

Submit the code. The log confirms that you have successfully connected to the CAS server.

Use the following caslib statement to access the CAS libraries within SAS 9.

```
caslib _all_ assign;
```

This statement will assign all of the CAS libraries that are available to your user ID and make them visible in SAS. In your environment, you might already have data loaded in your assigned caslibs. Because CAS processes only in-memory tables, we have to load tables into memory before we can use them in CAS. You will learn more about caslibs and loading data in the next chapter.

To terminate your CAS session, enter the following statement before exiting SAS Studio.

```
cas mysession terminate;
```

SAS Enterprise Guide Example

If you need to connect to CAS in Enterprise Guide 7.15 or SAS 9M5, let's see how easy it is to do that. When you open your Enterprise Guide session, it's important to make sure it has already been configured to access CAS. Submit the same statements in the previous section to start your CAS session from Enterprise Guide and access your caslibs.

```
casmySession sessopts=(caslib=casuser timeout=1000 locale=”en_US”);

caslib _all_ assign;
```

Regardless of which SAS programming interface you are using, you can now submit code to SAS9 or SAS Viya using SAS Studio, SAS Enterprise Guide, or the SAS Windowing Environment. In the next chapter, we will look at foundational programming in SAS Viya and the differences between SAS 9 code and SAS Viya code that takes advantage of CAS.

Resources

This chapter is based on the "Introduction to SAS Viya" videos in SAS® Viya® Enablement, a free course available from SAS Education.

You may find the following documentation helpful as you learn more about deployment of SAS® Viya®:

- SAS® Viya® 3.2: Deployment Guide

To stay informed about SAS Viya development, please refer to the SAS Viya Community website.

Chapter 2: Foundational Programming in SAS Viya

Introduction

The first thing you should know about SAS Viya is that you can use all of your existing SAS programming knowledge in this new, high-powered SAS environment. With your SAS experience and the capabilities of SAS Viya, you will be able to more efficiently and effectively analyze your data.

SAS offers a collection of new, high-performance CAS procedures as well as SAS procedures that will be familiar to users of SAS 9 and run in CAS with familiar syntax. The DATA step, DS2, and FedSQL all run in CAS as well. However, some aspects of the SAS programming language are not compatible with a multi-threaded approach. For example, you might want to run a DATA step in multiple threads in CAS and other times you need the DATA step to process the entire table sequentially on the same thread on either the CAS server or the workspace server. To address this, SAS Viya not only provides the CAS server, but also a SAS workspace server that is single-threaded so that you can choose.

There are three options for writing your programs in SAS Viya:

- **SAS Studio** provides a SAS programming environment for developing and submitting programs to the server.
- **Batch submission** is also still an option.
- **Open-source languages** such as Python, Lua, and Java can submit code to the CAS server.

In this chapter, you will learn how to access your data in SAS Viya, including how to load programs and access libraries. Then we will look at some simple DATA step programs to show how code in SAS Viya differs from SAS 9 code. We will also look at PROC FORMAT to show how to create and apply user-defined formats in SAS Viya. Finally, we will briefly look at Code Snippets, a feature in SAS Studio that allows you to access pre-defined code and save your own code for later use.

Accessing Data

Accessing your data is a critical part of any SAS program. It might not be the most exciting part of your data analysis, but you won't get far without it! In the previous chapter, we learned how to connect to the CAS server. In this section, we will look at some key concepts for accessing data in SAS Viya and then compare and contrast the code in SAS 9 and SAS Viya.

A Quick-Start Guide to Loading Data in CAS

Let's quickly learn how to load data into a caslib, view tables, and save them in-memory from SAS Studio. This is the first step to working with the data in SAS Viya instead of working with it locally. A caslib is a container for both the files in the caslib's data source and the in-memory tables that you load from the data source. We will discuss caslibs more in depth in the next section.

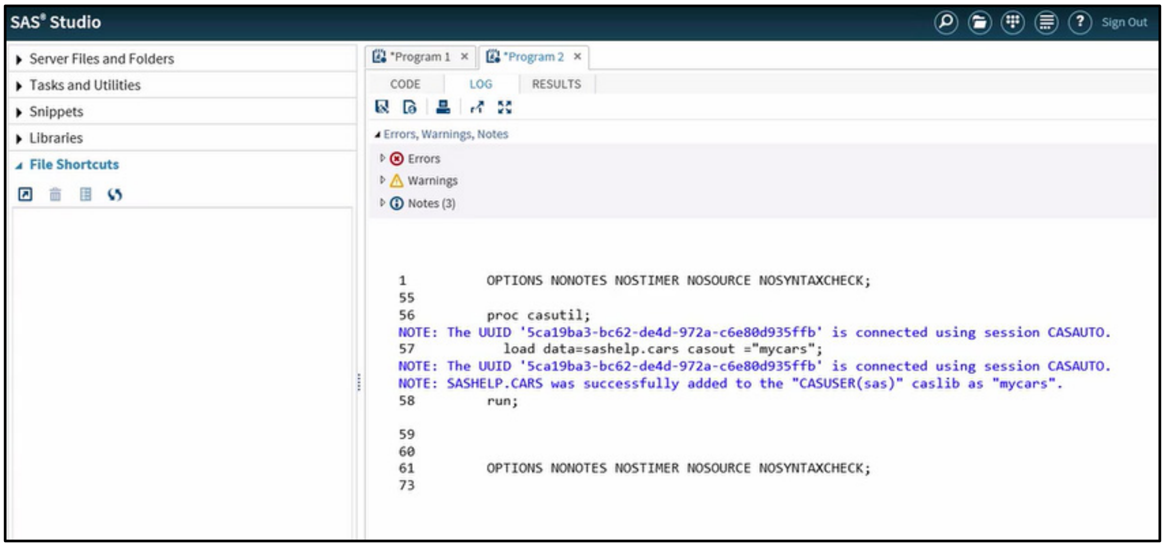
How to Load a SAS Table

Let's start with an example that loads a table from a SAS data set in SAS Studio. We will specify an alternative name to use for the loaded table by using the CASOUT= option, as shown in Program 2.1.

Program 2.1: Load SAS Table

```
proc casutil;
   load data=sashelp.cars casout=”mycars”;
run;
```

Let's look at the log. We can see that sashelp.cars was successfully added to my active caslib as MYCARS.



```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
55
56         proc casutil;
NOTE: The UUID '5ca19ba3-bc62-de4d-972a-c6e80d935ffb' is connected using session CASAUTO.
57            load data=sashelp.cars casout ="mycars";
NOTE: The UUID '5ca19ba3-bc62-de4d-972a-c6e80d935ffb' is connected using session CASAUTO.
NOTE: SASHELP.CARS was successfully added to the "CASUSER(sas)" caslib as "mycars".
58         run;

59
60
61         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
73
```

How to Load an Excel File

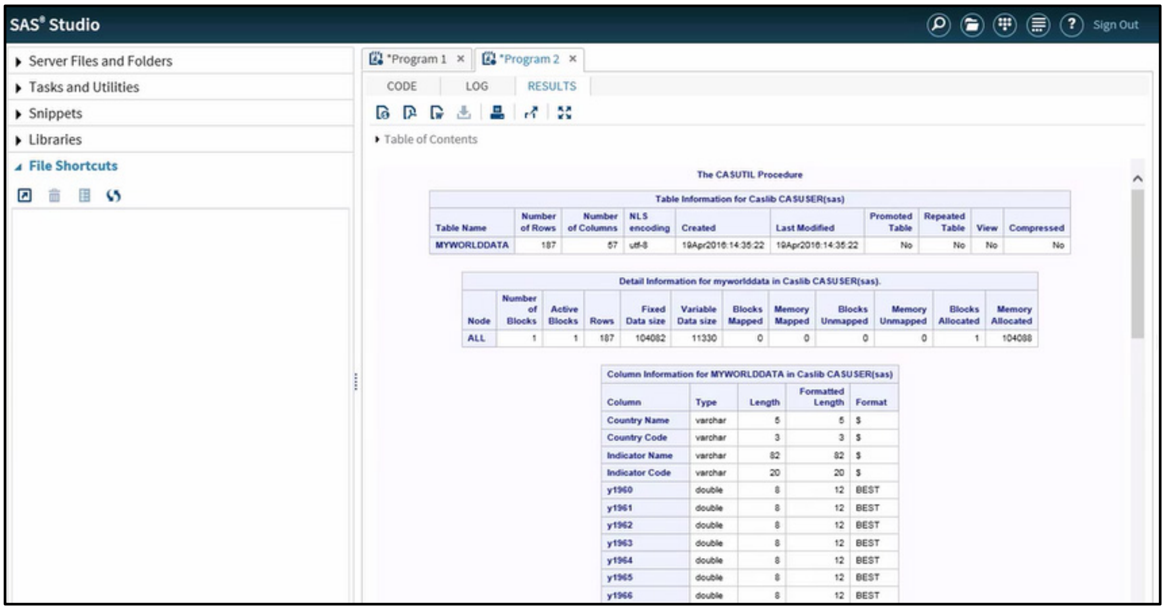
Next, let's read a spreadsheet. In Program 2.2, we will specify an alternative name to use for the loaded table. We will also run the CONTENTS statement to display metadata such as column names and data types to make sure that the Excel file is as we expect it to be.

Program 2.2: Load File

```
proc casutil;
   load file=”&datadir./WorldData.xlsx” casout=”myworlddata”;
   contents casdata=”myworlddata”;
run;
```

At the top of the results in Output 2.2, there is table information including memory allocation followed by column information for MYWORLDDATA. If you go to the log, you will see that a table, MYWORLDDATA has been created and has been loaded into your active caslib.

Output 2.2: Results of Program 2.2



The CASUTIL Procedure

Table Information for Caslib CASUSER(sas)

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
MYWORLDDATA	187	57	utf-8	19Apr2016:14:35:22	19Apr2016:14:35:22	No	No	No	No

Detail Information for myworlddata in Caslib CASUSER(sas).

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	1	1	187	104082	11330	0	0	0	0	1	104088

Column Information for MYWORLDDATA in Caslib CASUSER(sas)

Column	Type	Length	Formatted Length	Format
Country Name	varchar	5	5	$
Country Code	varchar	3	3	$
Indicator Name	varchar	82	82	$
Indicator Code	varchar	20	20	$
y1960	double	8	12	BEST
y1961	double	8	12	BEST
y1962	double	8	12	BEST
y1963	double	8	12	BEST
y1964	double	8	12	BEST
y1965	double	8	12	BEST
y1966	double	8	12	BEST

List Tables

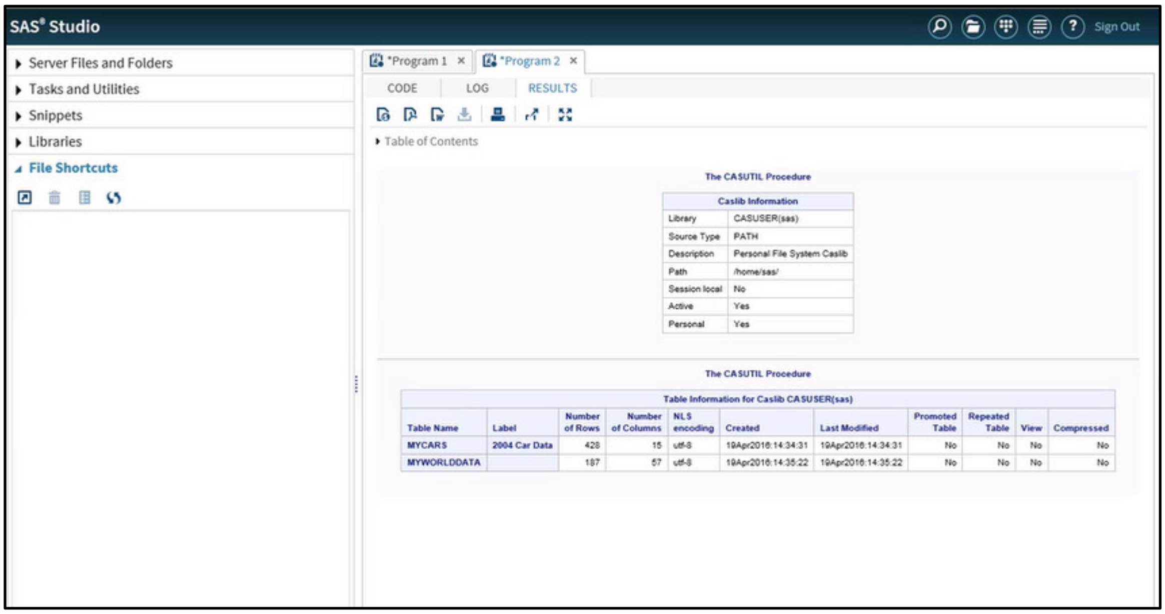
In the logs of both Programs 2.1 and 2.2 we can see that we have loaded both MYCARS and MYWORLDDATA. But there is another way to tell which tables have been loaded. We can use the LIST TABLES statement in Program 2.3 to list the in-memory tables.

Program 2.3: LIST TABLES

```
proc casutil;
    list tables;
run;
```

Let's look at the results in Output 2.2. Notice that the active caslib is CASUSER(viyauser). And the loaded tables are MYCARS and MYWORLDDATA, just as we would expect from this example. Both of these tables are available as CAS tables in your active caslib, and they will remain available until you end your CAS session.

Output 2.3: Results of Program 2.3



The CASUTIL Procedure

Caslib Information	
Library	CASUSER(sas)
Source Type	PATH
Description	Personal File System Caslib
Path	/home/sas/
Session local	No
Active	Yes
Personal	Yes

The CASUTIL Procedure

Table Information for Caslib CASUSER(sas)

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
MYCARS	2004 Car Data	428	15	utf-8	19Apr2016:14:34:31	19Apr2016:14:34:31	No	No	No	No
MYWORLDDATA		187	57	utf-8	19Apr2016:14:35:22	19Apr2016:14:35:22	No	No	No	No

Save Tables

An alternative way to save tables is shown in Program 2.4. You can use a SAVE statement to create a permanent copy of these tables saved to the data source associated with the caslib as SASHDAT files.

Program 2.4: SAVE statement

```
proc casutil;
   save casdata=”mycars”;
run;
```

Assign a Libref

Let’s learn how to associate a SAS libref with tables on the CAS server so that you can access them with procedure and DATA steps. In Program 2.5, the active caslib is used. After the code is run, the libref MYCAS will access the active caslib, and it is associated with the tables in the active caslib.

Program 2.5: LIBNAME statement

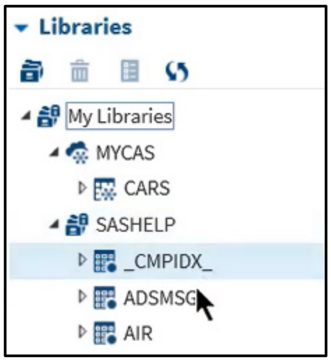
```
libname mycas cas;
```

In addition, you can add a file directory structure at the conclusion of the LIBNAME statement.

Accessing Libraries

If you are working in SAS Studio, you can go to the libraries section of the Navigation pane to expand the mycas library. You will notice in Figure 2.1 that there are slightly different icons that distinguish the caslib and the CAS tables from the SAS libraries and SAS data sets.

Figure 2.1: Libraries



Shut Down Active Sessions

When you finish working, you will want to shut down any active sessions. The code in Program 2.6 will help you determine the names of any active sessions.

Program 2.6: Determine Active Sessions

```
cas _all_ list;
```

In the log, you will see the names of any active sessions that need to be shut down. In this example, the name of the active session is CASAUTO, so we can shut down that active session using the code in Program 2.7.

Program 2.7: Terminate Active Sessions

```
cas casauto terminate;
```

Differences Between SAS 9 and SAS Viya

Now that you are familiar with some simple statements in SAS Viya, let’s consider a program in SAS 9 and look at how the code in SAS Viya differs.

In SAS 9, we use the familiar LIBNAME statement to establish a connection with the data. In the LIBNAME statement in Program 2.8, the libref orion creates a logical reference or "shortcut" to a physical location. Then we can take advantage of the orion libref throughout the rest of the program to identify the location of the tables we reference.

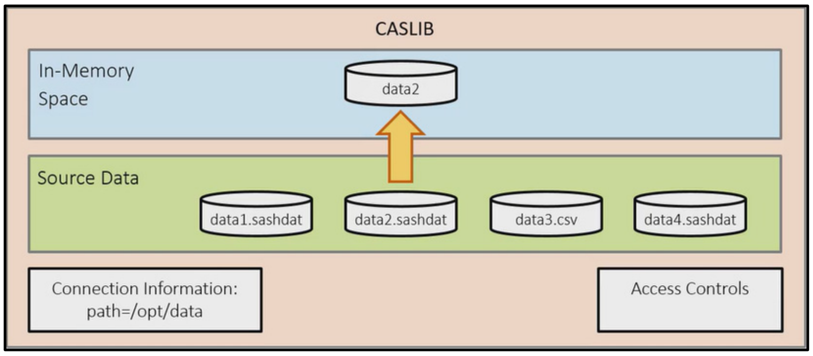
Program 2.8: SAS 9 code

```
libname orion "c:\mydata";

data orion.cars;
   set sashelp.cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   Keep Make Model Type MSRP Average_MPG;
run;
```

In SAS Viya, we also need to identify the data that we want to access and analyze with SAS Cloud Analytic Services (CAS). A caslib is how the CAS server accesses data. The best way to describe a caslib is that it is a container. The container has two areas where data is referenced: a physical space that includes the source data or files, and an in-memory space that makes the data available for CAS processing. The container also holds connection information for the source data and access controls governing who can access the caslib and what they can do with it.

To process data in CAS, the source data must be loaded into memory, as shown in Figure 2.2. The in-memory data is referred to as a CAS table. Additional tables can be loaded into memory from the caslib's data source or data can be loaded into the caslib's in-memory space from areas other than the caslib's source path. For example, we can load CSV, Excel, or SAS data sets into CAS. When we are finished with a CAS table, it can be dropped from the in-memory space. If we make changes to an in-memory table, they can be saved back to the caslib's data source so that the changes are available the next time you load the table to memory.

Figure 2.2: CASLIB



Now that we have illustrated the caslib conceptually, let's look at what this means for a SAS program. Program 2.9 shows how to load a table into CAS.

Program 2.9: SAS Viya code

```
cas mysess sessopts=(caslib=casuser);  ❶
libname mycas cas caslib=casuers;      ❷

proc casutil;                          ❸
   load data=sashelp.cars replace;
run;

data mycas.cars;
   set mycas.cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   Keep Make Model Type MSRP Average_MPG;
run;

proc casutil;
   save casdata="cars" replace;
   droptable casdata="cars";
run;
```

❶ Before we add a caslib, we need to start a CAS session. A CAS statement initiates a CAS session that in Program 2.9 is called mysess. The SESSOPTS= option is used with the CASLIB= session option to ensure that the casuser personal caslib is set as the active caslib. Much like the traditional sasuser library in SAS 9, casuser is your own personal caslib assigned to your credentials.

❷ Now we need to associate a libref with my casuser caslib to be able to use it in the program as libref.tablename. The libref will be mycas, and it will use the CAS engine and connect to the casuser caslib.

❸ If you have a SAS 9 data set that you want to process on the CAS server, you can use the new procedure PROC CASUTIL to load that table into memory in your caslib. In the LOAD statement, the DATA= option identifies the table that we want to load. The REPLACE option will overwrite the table if it already exists in the caslib.

At this point, we can take advantage of the DATA step or many other CAS-enabled procedures to manipulate or analyze our data. In the next section, we will continue working with the same program and look at the DATA step.

DATA Step

The familiar DATA step code that you use in SAS Version 9 can also work in SAS Viya. Compare the differences between the code in Programs 2.8 and 2.9 side by side below. Notice in Program 2.9 that we use standard SAS naming conventions to reference the data, with mycas as the libref and cars as the table.

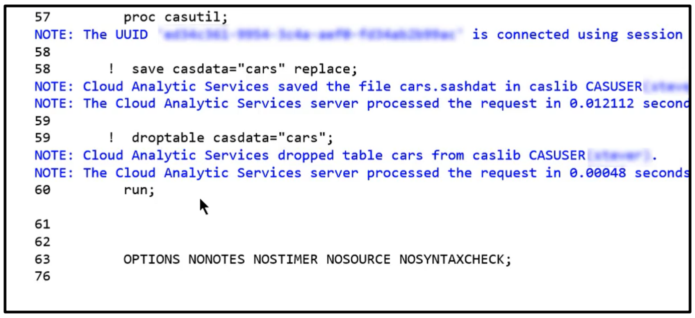
Program 2.8: SAS 9 code	**Program 2.9: SAS Viya code**
`libname orion “c:\mydata”;`	`cas mysess sessopts=(caslib=casuser);`
	`libname mycas cas caslib=casuers;`
	`proc casutil;`
	`  load data=sashelp.cars replace;`
	`run;`
`data orion.cars;`	`data mycas.cars;`
`  set sashelp.cars;`	`  set mycas.cars;`
`  Average_MPG=mean(MPG_City,`	`  Average_MPG=mean(MPG_City,`
`       MPG_Highway);`	`       MPG_Highway);`
`  Keep Make Model Type MSRP`	`  Keep Make Model Type MSRP`
`        Average_MPG;`	`       Average_MPG;`
`run;`	`run;`
	`proc casutil;`
	`      save casdata=”cars” replace;`
	`      droptable casdata=”cars”;`
	`run;`

After running the DATA step in Program 2.9, the log confirms that it ran in CAS. This is the case when both the DATA and SET statements reference a CAS table.

Saving Modified Tables

After modifying the cars CAS table in Program 2.9, we can save a physical copy in the source data space of the caslib. This can be done with PROC CASUTIL and the SAVE statement as shown in Program 2.9. The CASDATA= option names the CAS table to save. Once the physical table is created, we can drop the cars table from memory with the DROPTABLE statement.

If we run the final step in Program 2.9 and look at the log, we see that CAS saved cars.sashdat in the casuser caslib and that the cars table was dropped from memory.



```
57        proc casutil;
NOTE: The UUID '[redacted]' is connected using session
58
58      !  save casdata="cars" replace;
NOTE: Cloud Analytic Services saved the file cars.sashdat in caslib CASUSER[redacted]
NOTE: The Cloud Analytic Services server processed the request in 0.012112 second
59
59      !  droptable casdata="cars";
NOTE: Cloud Analytic Services dropped table cars from caslib CASUSER[redacted].
NOTE: The Cloud Analytic Services server processed the request in 0.00048 seconds
60        run;

61
62
63        OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
76
```

Differences Between SAS 9 and SAS Viya

Let's look at another example of the differences in the DATA step. Program 2.10 shows a simple SAS 9 program.

Program 2.10: SAS 9 code

```
data bigcars;
   set sashelp.cars;
   do i=1 to 1000000;
         output;
   end;
run;

data bigcars_score;
   set bigcars;
   length myscore 8;
   myscore=0.3*Invoice/(MSRP-Invoice)
         + 0.5*(EngineSize+Horsepower)/Weight + 0.2*(MPG_City+MPG_Highway);
run;
```

In the first DATA step, we are simply creating a large version of the sashelp.cars data set that has 42.8 million rows. In the second DATA step, we are creating a new variable named myscore that is a formula based on values within the data. If we run this program in SAS Studio with SAS 9.4 running behind the scenes, then the log will show that the scoring DATA step takes about 23 seconds real time and almost 11 seconds CPU time. Not bad for such a large table, but keep in mind this program is fairly simple data and code that ran on a single machine, single processor, on data that is stored on disk.

```
NOTE: There were 42800000 observations read from the data set WORK.BIGCARS.
NOTE: The data set WORK.BIGCARS_SCORE has 42800000 observations and 17 variables.
NOTE: DATA statement used (Total process time):
          real time             23.01 seconds
          cpu time              10.88 seconds
```

Now let's take advantage of SAS Viya's Cloud Analytics Services (CAS). When Program 2.10 is run in CAS, SAS takes advantage of multiple machines and processors in your cloud environment to load the data into memory and partition the DATA step execution out among multiple worker nodes. With each of these nodes running their piece of the DATA step in parallel, obviously this divide and conquer approach can mean dramatic performance improvements for complex programs and large data sets.

When we switch to a version of SAS Studio that is running SAS Viya behind the scenes, we need to add a few statements to the code to ensure it is running in CAS.

Program 2.11: CAS code

```
cas MySession sessopts=(caslib=casuser);         ❶
libname mycas cas caslib=casuser;

proc casutil;                                    ❷
   load data=sashelp.cars replace;
run;

data mycas.bigcars;
   set mycas.sashelp.cars;
   do i=1 to 1000000;
         output;
   end;
run;

data mycas.bigcars_score;
   set mycas.bigcars;
   length myscore 8;
   myscore=0.3*Invoice/(MSRP-Invoice)
         + 0.5*(EngineSize+Horsepower)/Weight + 0.2*(MPG_City+MPG_Highway);
   Thread=_threadid_;                            ❸

run;
```

❶ The first two statements start our CAS session then bind a traditional SAS libref, mycas, to a CAS library, which enables the loading and reading of data stored in memory. Note that the library references are updated in the rest of the program so that all data sets now reference the mycas library.

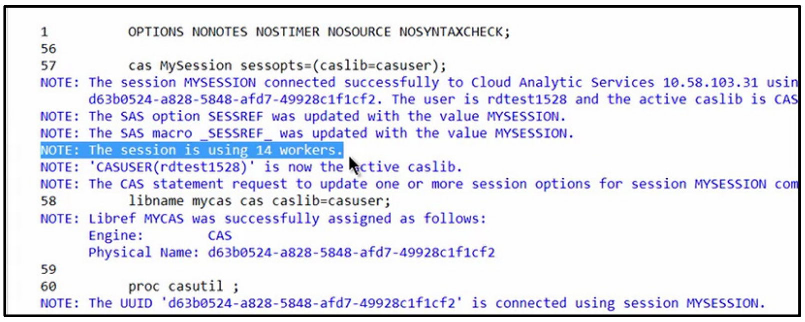
❷ The PROC CASUTIL statement loads the sashelp.cars table to the in-memory CAS library.

❸ The final addition is an assignment statement, creating a new column called Thread. This column will indicate on which thread each row was processed.

Again, we can compare the code side by side to see the differences.

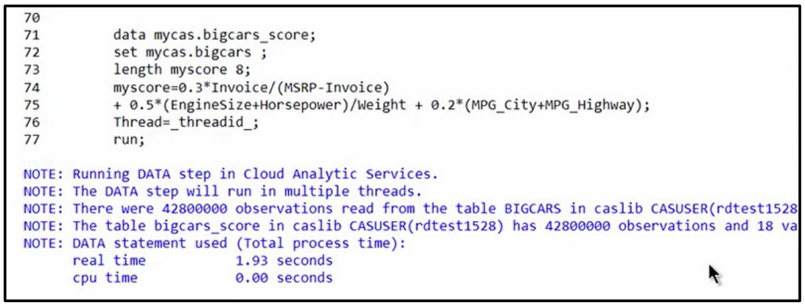
Program 2.10: SAS 9 code	Program 2.11: SAS Viya code
	`cas MySession`
	`sessopts=(caslib=casuser);`
	`libname mycas cas caslib=casuser;`
	`proc casutil;`
	`load data=sashelp.cars replace;`
	`run;`
`data bigcars;`	`data mycas.bigcars;`
`  set sashelp.cars;`	`  set mycas.sashelp.cars;`
`  do i=1 to 1000000;`	`  do i=1 to 1000000;`
`    output;`	`    output;`
`  end;`	`  end;`
`run;`	`run;`
`data bigcars_score;`	`data mycas.bigcars_score;`
`  set bigcars;`	`  set mycas.bigcars;`
`  length myscore 8;`	`  length myscore 8;`
`  myscore=0.3*Invoice/(MSRP-`	`  myscore=0.3*Invoice/(MSRP-`
`    Invoice) + 0.5*(EngineSize+`	`    Invoice) + 0.5*(EngineSize+`
`    Horsepower)/Weight +`	`    Horsepower)/Weight +`
`    0.2*(MPG_City+MPG_Highway);`	`    0.2*(MPG_City+MPG_Highway);`
`run;`	`  Thread=_threadid_;`
	`run;`

Let's run Program 2.11 and examine the log. We can see our CAS session has started and that we have 14 workers. Each worker has 16 CPU cores, which gives us a total of 224 threads.



```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
56
57         cas MySession sessopts=(caslib=casuser);
NOTE: The session MYSESSION connected successfully to Cloud Analytic Services 10.58.103.31 usin
      d63b0524-a828-5848-afd7-49928c1f1cf2. The user is rdtest1528 and the active caslib is CAS
NOTE: The SAS option SESSREF was updated with the value MYSESSION.
NOTE: The SAS macro _SESSREF_ was updated with the value MYSESSION.
NOTE: The session is using 14 workers.
NOTE: 'CASUSER(rdtest1528)' is now the active caslib.
NOTE: The CAS statement request to update one or more session options for session MYSESSION com
58         libname mycas cas caslib=casuser;
NOTE: Libref MYCAS was successfully assigned as follows:
      Engine:        CAS
      Physical Name: d63b0524-a828-5848-afd7-49928c1f1cf2
59
60         proc casutil ;
NOTE: The UUID 'd63b0524-a828-5848-afd7-49928c1f1cf2' is connected using session MYSESSION.
```

Now, looking at the lines of the log for the DATA step where the scoring took place, notice that it ran in 1.93 seconds real time and practically no CPU time. That’s quite an improvement!



```
70
71         data mycas.bigcars_score;
72         set mycas.bigcars ;
73         length myscore 8;
74         myscore=0.3*Invoice/(MSRP-Invoice)
75         + 0.5*(EngineSize+Horsepower)/Weight + 0.2*(MPG_City+MPG_Highway);
76         Thread=_threadid_;
77         run;

NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: There were 42800000 observations read from the table BIGCARS in caslib CASUSER(rdtest1528
NOTE: The table bigcars_score in caslib CASUSER(rdtest1528) has 42800000 observations and 18 va
NOTE: DATA statement used (Total process time):
      real time           1.93 seconds
      cpu time            0.00 seconds
```

This is just one example of the power and simplicity of SAS’ cloud analytics services for data manipulation with relatively simple data and code. Imagine what SAS Viya can do with more complex data and programs.

BY Statement for Processing Data in Groups

Using the BY statement in the DATA step to process data in groups is a powerful feature that programmers have long known and loved in SAS 9. For example, the BY statement allows us to use FIRST. and LAST. Variables, and is often necessary when merging. Let’s talk about how BY-group processing in the DATA step works in SAS Viya.

BY-group Processing in SAS 9

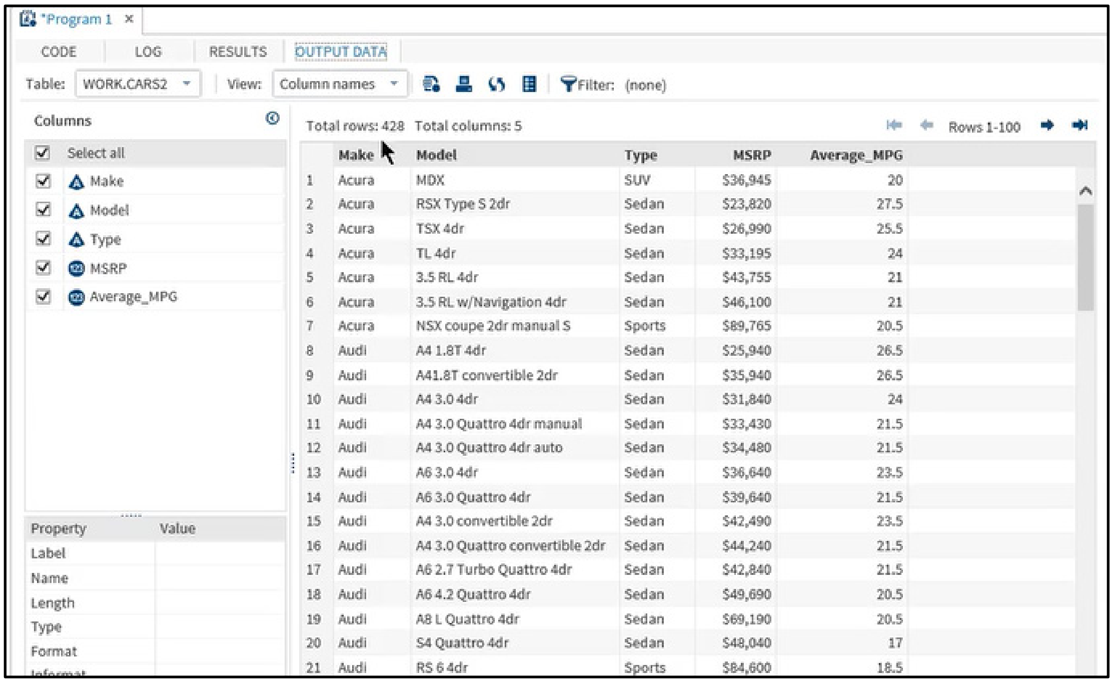
First, take a look at Program 2.12. It reads sashelp.cars, creates a new variable, and keeps only certain variables.

Program 2.12: SAS 9 code

```
data cars2;
   set sashelp.cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   keep Make Model Type Average_MPG MSRP;
run;
```

When Program 2.12 is submitted in SAS Viya, it runs on the SAS workspace server in a single thread – same code, same results as running it in SAS 9. The table viewer in Output 2.12 shows that cars2 is ordered by Make.

Output 2.12: Results of Program 2.12



Total rows: 428 Total columns: 5

	Make	Model	Type	MSRP	Average_MPG
1	Acura	MDX	SUV	$36,945	20
2	Acura	RSX Type S 2dr	Sedan	$23,820	27.5
3	Acura	TSX 4dr	Sedan	$26,990	25.5
4	Acura	TL 4dr	Sedan	$33,195	24
5	Acura	3.5 RL 4dr	Sedan	$43,755	21
6	Acura	3.5 RL w/Navigation 4dr	Sedan	$46,100	21
7	Acura	NSX coupe 2dr manual S	Sports	$89,765	20.5
8	Audi	A4 1.8T 4dr	Sedan	$25,940	26.5
9	Audi	A41.8T convertible 2dr	Sedan	$35,940	26.5
10	Audi	A4 3.0 4dr	Sedan	$31,840	24
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	$33,430	21.5
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	$34,480	21.5
13	Audi	A6 3.0 4dr	Sedan	$36,640	23.5
14	Audi	A6 3.0 Quattro 4dr	Sedan	$39,640	21.5
15	Audi	A4 3.0 convertible 2dr	Sedan	$42,490	23.5
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	$44,240	21.5
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	$42,840	21.5
18	Audi	A6 4.2 Quattro 4dr	Sedan	$49,690	20.5
19	Audi	A8 L Quattro 4dr	Sedan	$69,190	20.5
20	Audi	S4 Quattro 4dr	Sedan	$48,040	17
21	Audi	RS 6 4dr	Sports	$84,600	18.5

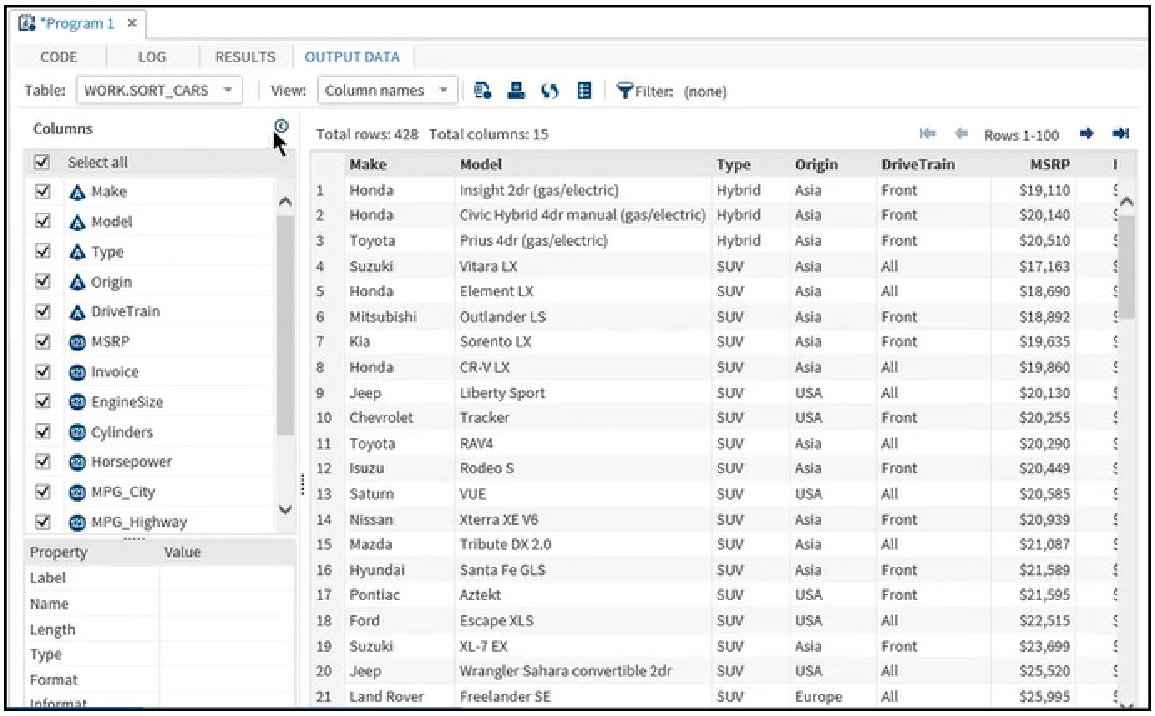
What if instead we would like to sort by Type, and then sort within Type by MSRP? Of course, that requires a PROC SORT step before the DATA step. We will use first. and last. variables to flag the high and low values of MSRP within Type. We add the BY statement in the DATA step to create the first./last. variables, and conditional logic to assign values to our new variables, as shown in Program 2.13.

Program 2.13: Modified SAS 9 code

```
proc sort data=sashelp.cars out=sort_cars;
   by Type MSRP;
run;

data cars2;
   set sort_cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   keep Make Model Type Average_MPG MSRP LowMSRP HighMSRP;
   by Type;
   if first.Type then LowMSRP=1;
         else LowMSRP=0;
   if last.Type then HighMSRP=1;
         else HighMSRP=0;
run;
```

Again, when we run this code in SAS Viya, the DATA step is running in SAS, single-threaded, with all the rows being processed sequentially, just as in SAS 9. See Output 2.13.

Output 2.13: Results of Program 2.13



Total rows: 428 Total columns: 15

	Make	Model	Type	Origin	DriveTrain	MSRP
1	Honda	Insight 2dr (gas/electric)	Hybrid	Asia	Front	$19,110
2	Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	Asia	Front	$20,140
3	Toyota	Prius 4dr (gas/electric)	Hybrid	Asia	Front	$20,510
4	Suzuki	Vitara LX	SUV	Asia	All	$17,163
5	Honda	Element LX	SUV	Asia	All	$18,690
6	Mitsubishi	Outlander LS	SUV	Asia	Front	$18,892
7	Kia	Sorento LX	SUV	Asia	Front	$19,635
8	Honda	CR-V LX	SUV	Asia	All	$19,860
9	Jeep	Liberty Sport	SUV	USA	All	$20,130
10	Chevrolet	Tracker	SUV	USA	Front	$20,255
11	Toyota	RAV4	SUV	Asia	All	$20,290
12	Isuzu	Rodeo S	SUV	Asia	Front	$20,449
13	Saturn	VUE	SUV	USA	All	$20,585
14	Nissan	Xterra XE V6	SUV	Asia	Front	$20,939
15	Mazda	Tribute DX 2.0	SUV	Asia	All	$21,087
16	Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589
17	Pontiac	Aztekt	SUV	USA	Front	$21,595
18	Ford	Escape XLS	SUV	USA	All	$22,515
19	Suzuki	XL-7 EX	SUV	Asia	Front	$23,699
20	Jeep	Wrangler Sahara convertible 2dr	SUV	USA	All	$25,520
21	Land Rover	Freelander SE	SUV	Europe	All	$25,995

BY-group Processing in SAS Viya

When the DATA step runs on a CAS server, the default behavior is for CAS to distribute the input table across multiple machines and threads, based on the original sort order of the input table. But when we use the BY statement in the DATA step, the rows are grouped based on the first BY variable, and then distributed across multiple threads. In this way, rows within BY groups share the same server node. If a format is applied to the grouping variable, the groups are based on the formatted values. Here is the big difference from SAS 9: no prior PROC SORT is required!

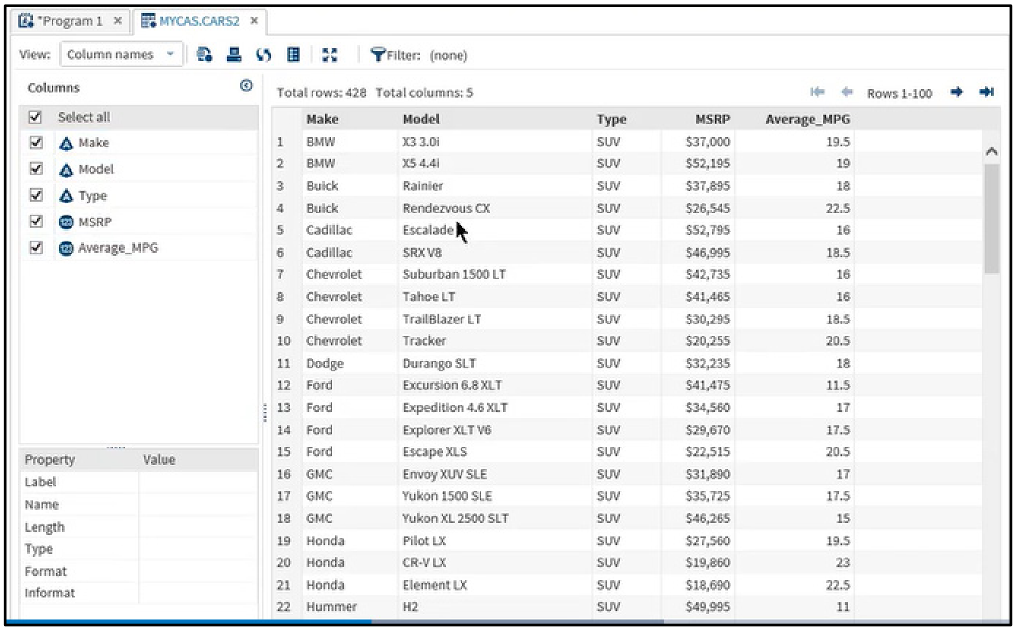
Notice that Program 2.14 does not include a PROC SORT.

Program 2.14: SAS Viya Code

```
cas mysess sessopts=(caslib=casuser);
libname mycas cas;

proc casutil;
   load data=sashelp.cars replace;
run;

data mycas.cars2;
   set mycas.cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   keep Make Model Type Average_MPG MSRP;
   by Type;
run;
```

When we run Program 2.14, we can see that the data is grouped by Type, but not in sorted sequence. Hybrid is toward the bottom of the data.

Output 2.14: Results of Program 2.14



Total rows: 428 Total columns: 5

	Make	Model	Type	MSRP	Average_MPG
1	BMW	X3 3.0i	SUV	$37,000	19.5
2	BMW	X5 4.4i	SUV	$52,195	19
3	Buick	Rainier	SUV	$37,895	18
4	Buick	Rendezvous CX	SUV	$26,545	22.5
5	Cadillac	Escalade	SUV	$52,795	16
6	Cadillac	SRX V8	SUV	$46,995	18.5
7	Chevrolet	Suburban 1500 LT	SUV	$42,735	16
8	Chevrolet	Tahoe LT	SUV	$41,465	16
9	Chevrolet	TrailBlazer LT	SUV	$30,295	18.5
10	Chevrolet	Tracker	SUV	$20,255	20.5
11	Dodge	Durango SLT	SUV	$32,235	18
12	Ford	Excursion 6.8 XLT	SUV	$41,475	11.5
13	Ford	Expedition 4.6 XLT	SUV	$34,560	17
14	Ford	Explorer XLT V6	SUV	$29,670	17.5
15	Ford	Escape XLS	SUV	$22,515	20.5
16	GMC	Envoy XUV SLE	SUV	$31,890	17
17	GMC	Yukon 1500 SLE	SUV	$35,725	17.5
18	GMC	Yukon XL 2500 SLT	SUV	$46,265	15
19	Honda	Pilot LX	SUV	$27,560	19.5
20	Honda	CR-V LX	SUV	$19,860	23
21	Honda	Element LX	SUV	$18,690	22.5
22	Hummer	H2	SUV	$49,995	11

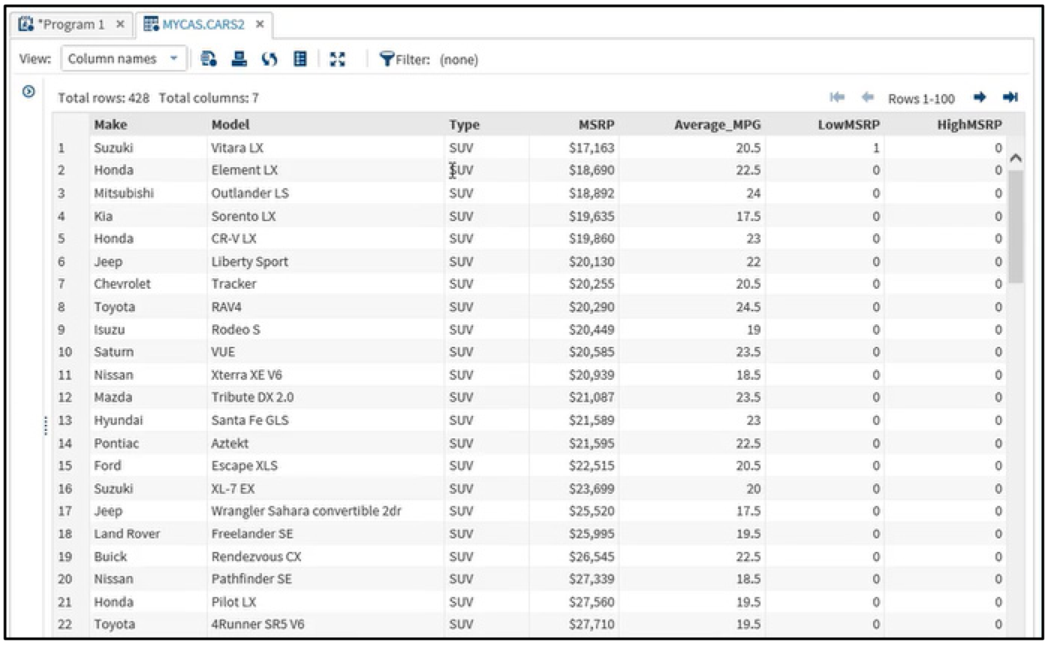
Remember in the SAS 9 Program 2.13 we sorted the data within Type by MSRP. One way to accomplish this is simply to add MSRP to the BY statement, as shown in Program 2.15. We will also include the statements that use first. and last. variables to create our new high and low MSRP columns.

Program 2.15: Modified SAS Viya code

```
data mycas.cars2;
   set mycas.cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   keep Make Model Type Average_MPG MSRP LowMSRP HighMSRP;
   by Type MSRP;
   if first.Type then LowMSRP=1;
         else LowMSRP=0;
   if last.Type then HighMSRP=1;
         else HighMSRP=0;
run;
```

After running the program, we see in Output 2.15 that the data is grouped by Type, although not in sorted order, but MSRP is sorted within each group so that the low and high MSRP columns are created correctly. When more than one variable is listed in the BY statement, CAS groups and distributes the rows based on only the values of the first BY variable, and then orders the data within the groups by any subsequent variables.

Output 2.15: Results of Program 2.15



Total rows: 428 Total columns: 7

	Make	Model	Type	MSRP	Average_MPG	LowMSRP	HighMSRP
1	Suzuki	Vitara LX	SUV	$17,163	20.5	1	0
2	Honda	Element LX	SUV	$18,690	22.5	0	0
3	Mitsubishi	Outlander LS	SUV	$18,892	24	0	0
4	Kia	Sorento LX	SUV	$19,635	17.5	0	0
5	Honda	CR-V LX	SUV	$19,860	23	0	0
6	Jeep	Liberty Sport	SUV	$20,130	22	0	0
7	Chevrolet	Tracker	SUV	$20,255	20.5	0	0
8	Toyota	RAV4	SUV	$20,290	24.5	0	0
9	Isuzu	Rodeo S	SUV	$20,449	19	0	0
10	Saturn	VUE	SUV	$20,585	23.5	0	0
11	Nissan	Xterra XE V6	SUV	$20,939	18.5	0	0
12	Mazda	Tribute DX 2.0	SUV	$21,087	23.5	0	0
13	Hyundai	Santa Fe GLS	SUV	$21,589	23	0	0
14	Pontiac	Aztekt	SUV	$21,595	22.5	0	0
15	Ford	Escape XLS	SUV	$22,515	20.5	0	0
16	Suzuki	XL-7 EX	SUV	$23,699	20	0	0
17	Jeep	Wrangler Sahara convertible 2dr	SUV	$25,520	17.5	0	0
18	Land Rover	Freelander SE	SUV	$25,995	19.5	0	0
19	Buick	Rendezvous CX	SUV	$26,545	22.5	0	0
20	Nissan	Pathfinder SE	SUV	$27,339	18.5	0	0
21	Honda	Pilot LX	SUV	$27,560	19.5	0	0
22	Toyota	4Runner SR5 V6	SUV	$27,710	19.5	0	0

PARTITION= and ORDERBY=

If we want to group the data and view it in sorted sequence, there are two new data set options that can be used for output CAS tables:

- PARTITION=
- ORDERBY=

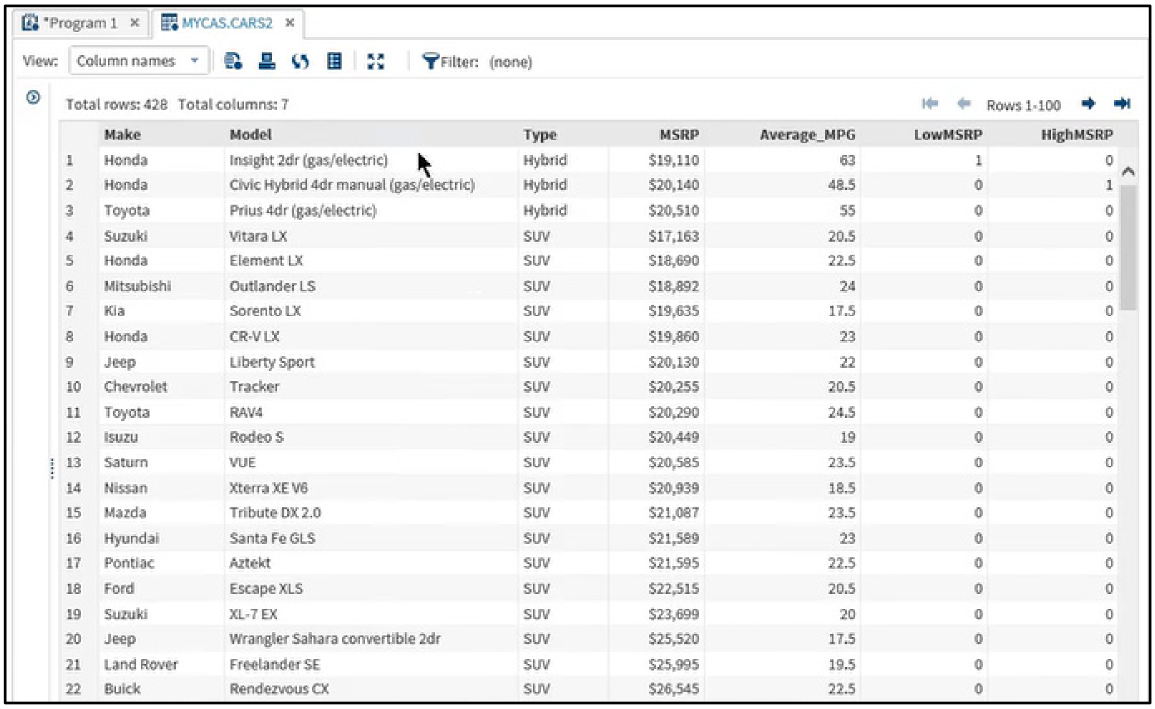
The PARTITION= data set option specifies one or more partitioning variables. CAS then uses the formatted values of the partitioning variables to divide and distribute the rows of the input table across threads. The ORDERBY= data set option orders the rows within each partition. See Program 2.16 for an example of code using these options.

Program 2.16: SAS Viya Code with PARTITION= and ORDERBY= Options

```
data mycas.cars2 (partition=(type) orderby=(MSRP));
   set mycas.cars;
   Average_MPG=mean(MPG_City, MPG_Highway);
   keep Make Model Type Average_MPG MSRP LowMSRP HighMSRP;
   by Type;
   if first.Type then LowMSRP=1;
         else LowMSRP=0;
   if last.Type then HighMSRP=1;
         else HighMSRP=0;
run;
```

Although the PARTITION= option is used in Program 2.16, the BY statement is still required to create the first. and last. variables. When we run Program 2.16, notice in Output 2.16 that the values of the PARTITION variable, Type, are in sorted sequence and within each Type, MSRP is sorted as well.

Output 2.16: Results of Program 2.16



Total rows: 428 Total columns: 7

	Make	Model	Type	MSRP	Average_MPG	LowMSRP	HighMSRP
1	Honda	Insight 2dr (gas/electric)	Hybrid	$19,110	63	1	0
2	Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	$20,140	48.5	0	1
3	Toyota	Prius 4dr (gas/electric)	Hybrid	$20,510	55	0	0
4	Suzuki	Vitara LX	SUV	$17,163	20.5	0	0
5	Honda	Element LX	SUV	$18,690	22.5	0	0
6	Mitsubishi	Outlander LS	SUV	$18,892	24	0	0
7	Kia	Sorento LX	SUV	$19,635	17.5	0	0
8	Honda	CR-V LX	SUV	$19,860	23	0	0
9	Jeep	Liberty Sport	SUV	$20,130	22	0	0
10	Chevrolet	Tracker	SUV	$20,255	20.5	0	0
11	Toyota	RAV4	SUV	$20,290	24.5	0	0
12	Isuzu	Rodeo S	SUV	$20,449	19	0	0
13	Saturn	VUE	SUV	$20,585	23.5	0	0
14	Nissan	Xterra XE V6	SUV	$20,939	18.5	0	0
15	Mazda	Tribute DX 2.0	SUV	$21,087	23.5	0	0
16	Hyundai	Santa Fe GLS	SUV	$21,589	23	0	0
17	Pontiac	Aztekt	SUV	$21,595	22.5	0	0
18	Ford	Escape XLS	SUV	$22,515	20.5	0	0
19	Suzuki	XL-7 EX	SUV	$23,699	20	0	0
20	Jeep	Wrangler Sahara convertible 2dr	SUV	$25,520	17.5	0	0
21	Land Rover	Freelander SE	SUV	$25,995	19.5	0	0
22	Buick	Rendezvous CX	SUV	$26,545	22.5	0	0

We have seen that SAS Viya uses practically the same syntax as SAS 9 to process data in groups using the BY statement in the DATA step. But given the fact that SAS Viya allows us to run the DATA step in CAS, we can potentially realize great gains in efficiency.

FORMAT Procedure

SAS formats have always provided a powerful way to look at your data in different ways. SAS offers a rich collection of formats and the FORMAT procedure allows you to create custom formats. In this section, we will look at how to create and apply user-defined formats in SAS Viya.

Formats in SAS 9

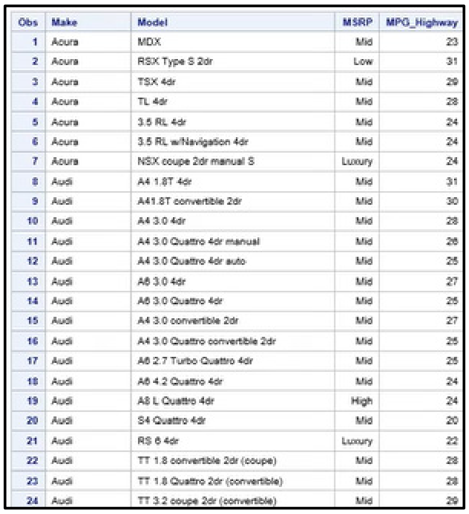
Let's start by looking at code that works in SAS 9. In Program 2.17, we use PROC FORMAT to create a user-defined format named PRICERANGE_SAS that groups the cost of cars into four categories. Creating a format won't do much good until we apply it, which is done by the FORMAT statement in the DATA step that follows.

Program 2.17: SAS 9 Code

```
proc format;
   value pricerange_sas low-25000=”Low”
                        25000<-50000=”Mid”
                        50000<-75000=”High”
                        75000<-high=”Luxury”;
run;

data cars_formatted;
   set sashelp.cars;
   format MSRP pricerange_sas.;
   keep Make Model MSRP MPG_Highway;
run;

proc print data=cars_formatted;
run;
```

After running Program 2.17, we see MSRP displayed as the formatted values in Output 2.17.

Output 2.17: Results of Program 2.17



Obs	Make	Model	MSRP	MPG_Highway
1	Acura	MDX	Mid	23
2	Acura	RSX Type S 2dr	Low	31
3	Acura	TSX 4dr	Mid	29
4	Acura	TL 4dr	Mid	28
5	Acura	3.5 RL 4dr	Mid	24
6	Acura	3.5 RL w/Navigation 4dr	Mid	24
7	Acura	NSX coupe 2dr manual S	Luxury	24
8	Audi	A4 1.8T 4dr	Mid	31
9	Audi	A41.8T convertible 2dr	Mid	30
10	Audi	A4 3.0 4dr	Mid	28
11	Audi	A4 3.0 Quattro 4dr manual	Mid	26
12	Audi	A4 3.0 Quattro 4dr auto	Mid	25
13	Audi	A6 3.0 4dr	Mid	27
14	Audi	A6 3.0 Quattro 4dr	Mid	25
15	Audi	A4 3.0 convertible 2dr	Mid	27
16	Audi	A4 3.0 Quattro convertible 2dr	Mid	25
17	Audi	A6 2.7 Turbo Quattro 4dr	Mid	25
18	Audi	A6 4.2 Quattro 4dr	Mid	24
19	Audi	A8 L Quattro 4dr	High	24
20	Audi	S4 Quattro 4dr	Mid	20
21	Audi	RS 6 4dr	Luxury	22
22	Audi	TT 1.8 convertible 2dr (coupe)	Mid	28
23	Audi	TT 1.8 Quattro 2dr (convertible)	Mid	28
24	Audi	TT 3.2 coupe 2dr (convertible)	Mid	29

In the SAS 9 world, formats are stored in catalogs in SAS libraries. In Program 2.17, the PRICERANGE_SAS format was saved in the default WORK library and FORMATS catalog.

Formats in SAS Viya

Remember, in SAS Viya, code and data can be processed in two different places: the SAS workspace server or the CAS server. When we apply a format to a non-CAS table in a step that is executed by the SAS workspace server, that format must also be available to the workspace server. In Program 2.17, all the steps and data were processed by the workspace server. Therefore, the format was found in the WORK.FORMATS catalog and applied to the data and PROC PRINT output .

CAS-enabled procedures can also take advantage of user-defined formats. But when the code and data run in CAS, the user-defined formats must be available via a CAS format library. Program 2.18 shows how the code will change when creating and applying formats to in-memory CAS tables. The end goal is to use a CAS procedure PROC MDSUMMARY to compute summary statistics on an in-memory table for each formatted value of MSRP.

Program 2.18

```
cas mysess sessopts=(caslib=casuser);   ❶
libname mycas cas;

proc format casfmtlib='casformats';     ❷
   value pricerange_cas low-25000="Low"
                        25000<-50000="Mid"
                        50000<-75000="High"
                        75000<-high="Luxury";
run;

data mycas.cars_formatted;        ❸
   set sashelp.cars;
   format MSRP pricerange_cas.;
   keep Make Model MSRP MPG_Highway;
run;

proc mdsummary data=mycas.cars_formatted;
   var MPG_Highway;
   groupby MSRP / out=mycas.cars_summary;
run;
```

❶ This code starts the CAS session and assigns a libref, to your caslib.

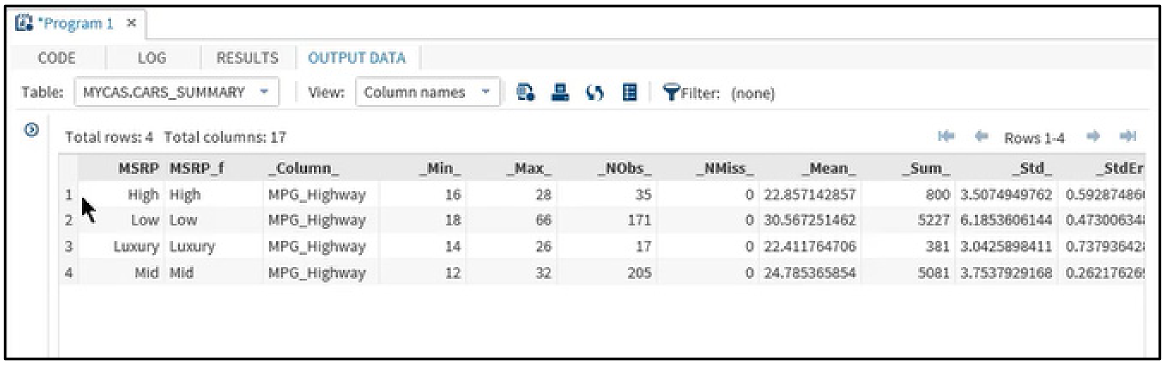
❷ Here we use PROC FORMAT to build the custom format, PRICERANGE_CAS. The only difference between this step and the SAS 9 example in Program 2.18 is the addition of the CASFMTLIB= option.

This option is used to add the format to a CAS format library named casformats. Now this PRICERANGE_SAS format will be available in the CAS session to use with in-memory data.

❸ In previous programs (Program 2.11 and 2.14), we used the CASUTIL procedure to load data in CAS. This time we are using an alternative method – the DATA step. In the DATA step, we use a FORMAT statement to apply the PRICERANGE_CAS format to MSRP. Notice that this DATA step is exactly the same as Program 2.17 with one small but significant difference: the DATA statement names a CAS table as the output source. So this step executes on the SAS workspace server, reads the sashelp.cars table from disk, assigns a CAS format, and loads the data in CAS. In future steps when this cars_formatted CAS table is referenced, the assigned PRICERANGE_CAS format can be pulled from the CAS format library.

The results of Program 2.18 are shown in Output 2.18. The cars_summary table created by PROC MDSUMMARY shows the statistics based on the formatted values of MSRP.

Output 2.18: Results of Program 2.18



*Program 1 ×

CODE | LOG | RESULTS | OUTPUT DATA

Table: MYCAS.CARS_SUMMARY   View: Column names   Filter: (none)

Total rows: 4 Total columns: 17   Rows 1-4

	MSRP	MSRP_f	_Column_	_Min_	_Max_	_NObs_	_NMiss_	_Mean_	_Sum_	_Std_	_StdEr
1	High	High	MPG_Highway	16	28	35	0	22.857142857	800	3.5074949762	0.59287486
2	Low	Low	MPG_Highway	18	66	171	0	30.567251462	5227	6.1853606144	0.47300634
3	Luxury	Luxury	MPG_Highway	14	26	17	0	22.411764706	381	3.0425898411	0.73793642
4	Mid	Mid	MPG_Highway	12	32	205	0	24.785365854	5081	3.7537929168	0.26217626

The notes in the log also confirm that MDSUMMARY was processed by CAS. If we returned to Program 2.18 and changed the format to PRICRANGE_SAS, the formatted created by the SAS workspace server, instead of PRICRANGE_CAS, the log would show that the DATA step ran successfully on the workspace server. But PROC MDSUMMARY would fail because CAS cannot load the PRICERANGE_SAS format. Steps that run in CAS must reference formats in a CAS format library.

The PRICERANGE_CAS format that we created is temporary and will persist only for the duration of our CAS session. You can save permanent formats for personal use or promote formats so that they can be used by other people or sessions.

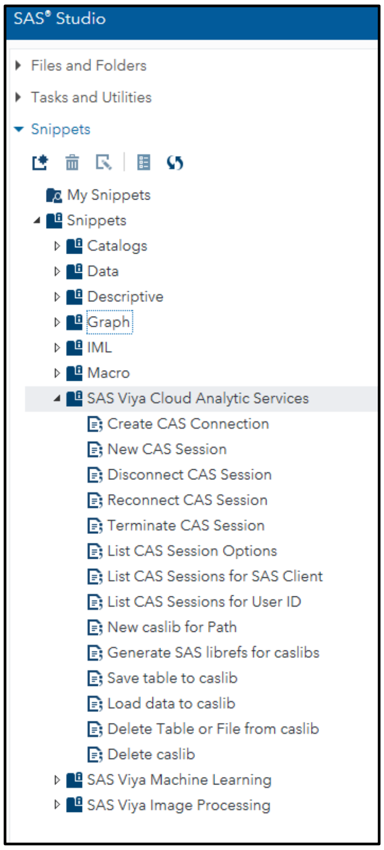
Code Snippets

Has there ever been a block of code that you use so infrequently that you always seem to forget the options that you need? Conversely, has there ever been a block of code that you use so frequently that you grow tired of typing it all the time? Code snippets can greatly assist with both of these scenarios. In this section, we discuss using pre-installed code snippets and creating new code snippets within SAS Viya.

Pre-installed Code Snippets

SAS Viya comes with several code snippets pre-installed. Recall that we used a snippet in the previous chapter to connect to CAS. To access these snippets, expand the Snippets area on the left navigation panel as shown in Figure 2.3. You can see that the snippets are divided into categories, making it easier to find them.

Figure 2.3: Pre-installed Snippets



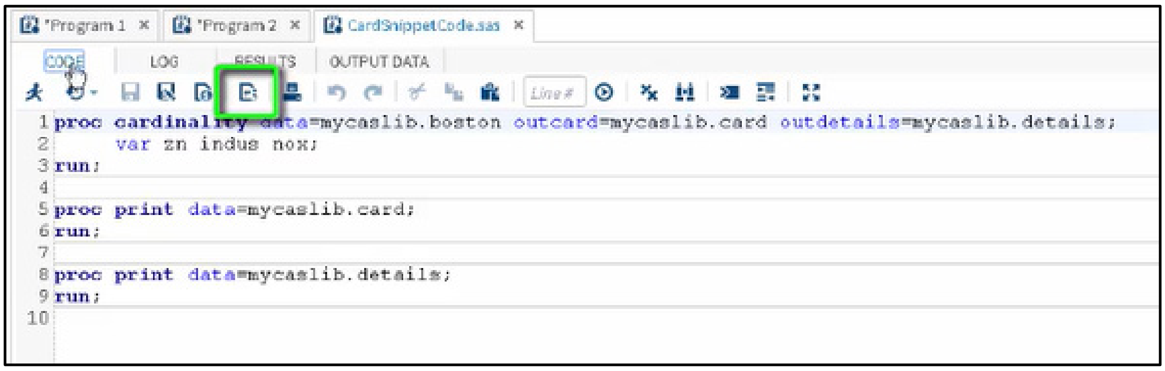
If you double-click a pre-installed code snippet, or if you click and drag the snippet into the code editor panel, then the snippet will appear in the panel.

Snippets can range from very simple to very complex. Some contain comments. Some contain macro variables. Some might be only a couple of lines of code. That is the advantage of snippets. They can be anything that you want them to be.

Create New Snippets

Now, let's create a snippet of our own. Figure 2.4 shows an example of code that calls PROC CARDINALITY. This code is complete and fully executable. When you have the code the way that you want in your code window, click on the shortcut button for Add to My Snippets above the code. The button is outlined in a box in Figure 2.4.

Figure 2.4: Add to My Snippets Button



```
proc cardinality data=mycaslib.boston outcard=mycaslib.card outdetails=mycaslib.details;
     var zn indus nox;
run;

proc print data=mycaslib.card;
run;

proc print data=mycaslib.details;
run;
```

A window will appear that asks you to name the snippet. Naming the snippet then saves it into the My Snippets area in the left navigation panel for future use.

Remember that snippets are extremely flexible. The code that you save does not have to be fully executable. Instead of supplying the data source in your code, you may instead include notes or comments about what needs to be added, which makes the code more general, but it is still a very useful snippet.

To use one of your saved snippets, simply navigate to the My Snippets area, then double-click on your snippet or drag it into the code window.

Resources

This chapter is based on the "An Introduction to SAS Viya Programming for SAS 9 Programmers" videos in "SAS® Viya® Enablement," a free course available from SAS Education.

You may find the following documentation helpful as you learn more about programming in SAS® Viya®:

- SAS® Viya® 3.2 Programming Quick Start
  - An Introduction to SAS® Viya® 3.2 Programming
  - SAS® Viya® 3.2 Formats and Informats: Reference

Chapter 3: Statistical Programming in SAS Viya

Introduction

As you have seen in the previous chapter, you can use all of your existing SAS programming knowledge in SAS Viya to more efficiently and effectively analyze your data. SAS Viya also offers several new procedures that are available for sites with SAS Viya installed. You can see a complete list of the procedures with explanations in the documentation.

In this chapter, we will look at several new procedures that are available in SAS Viya to aid in your data wrangling and statistical programming. At the same time, we will look at the tasks in SAS Studio that can help you code some of these procedures and compare the features available in SAS Viya with the features of procedures you might already be using in SAS 9.

A predefined task in SAS Viya is an XML and Apache Velocity code file that generates SAS code and formats results for you. Tasks include SAS procedures from simple data listings to complex analytical procedures. You might already be familiar with tasks in SAS Studio that use SAS 9. If you license SAS Visual Analytics, SAS Visual Data Mining and Machine Learning, or any other SAS Viya products, you will have access to certain predefined tasks in SAS Studio.

This chapter is by no means a comprehensive guide to all of the new procedures, tasks, or features in SAS Viya. It is a gentle introduction to using SAS Viya for your statistical programming so that you can begin to take advantage of the scalability, elasticity, and flexibility of this modern analytics platform.

Prepare and Explore

In this section, you will learn how to use SAS Viya to perform data wrangling tasks including data exploration, sampling, and partitioning. You will also learn how to perform data transformation tasks such as variable imputation and variable binning.

Here are the procedures and SAS Studio tasks that will be discussed in this section:

Name	SAS Studio Task	SAS Viya Procedure	SAS 9 Procedure
Variable cardinality analysis	Summary	PROC CARDINALITY	
Sampling	Sampling	PROC PARTITION	

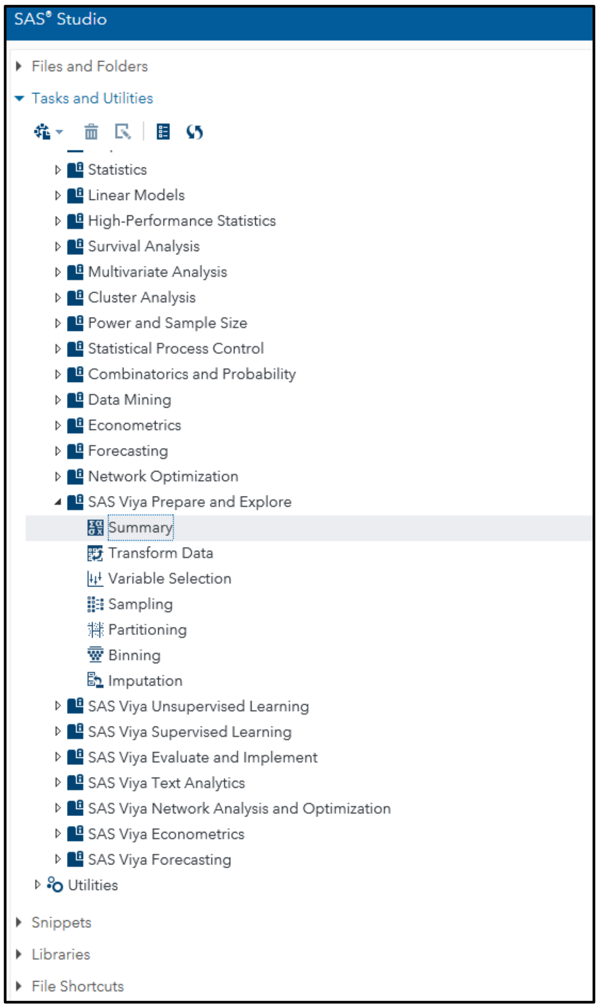
Name	SAS Studio Task	SAS Viya Procedure	SAS 9 Procedure
Partitioning	Partitioning	PROC PARTITION	
Missing value imputation	Imputation	PROC VARIMPUTE	
Variable Binning	Binning	PROC BINNING	
Variable Selection	Variable Selection	PROC VARREDUCE	PROC VARCLUS

The data sourced used in the examples contains a list of donors to an organization and has a target variable, TARGET_B, which is a binary variable that has the value 1 for a person who has donated during a mailing. The example data is available to download from the SAS Enterprise Miner 5.3 documentation at this link.

Data Exploration

To begin, in SAS Studio click on Tasks and Utilities on the left menu. Expand the Tasks and Prepare and Explore folders, as shown in Figure 3.1. Your menu may look different depending in which SAS Viya products your site has installed.

Figure 3.1: Prepare and Explore Folder

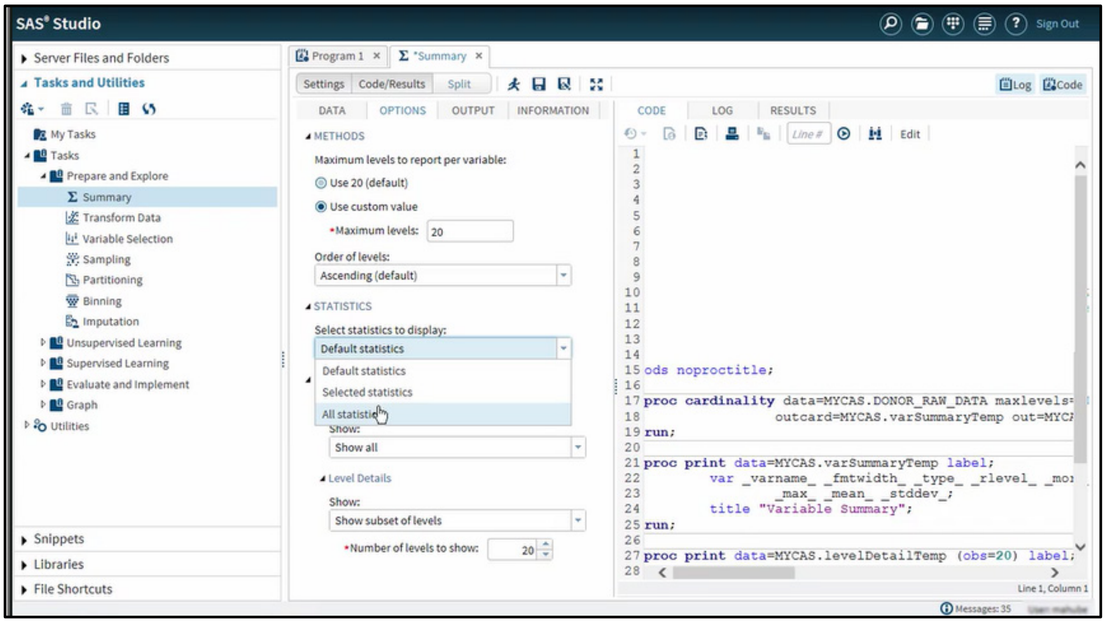


Summary

First, we will extract variables summary information from our data. To accomplish this, we will use the Summary task. Double click Summary from the Prepare and Explore folder in the left menu.

Then, select your data in the Data tab that opens in the Summary window. On the Options tab, we will select Use custom value and leave the default at 20. Under Statistics, select All Statistics, as shown in Figure 3.2.

Figure 3.2: Summary Options tab



METHODS

Maximum levels to report per variable:

Use 20 (default) [✓ Use custom value]

*Maximum levels: 20

Order of levels: Ascending (default)

STATISTICS

Select statistics to display: Default statistics

Default statistics

Selected statistics

All statistics

Show: Show all

Level Details

Show: Show subset of levels

*Number of levels to show: 20

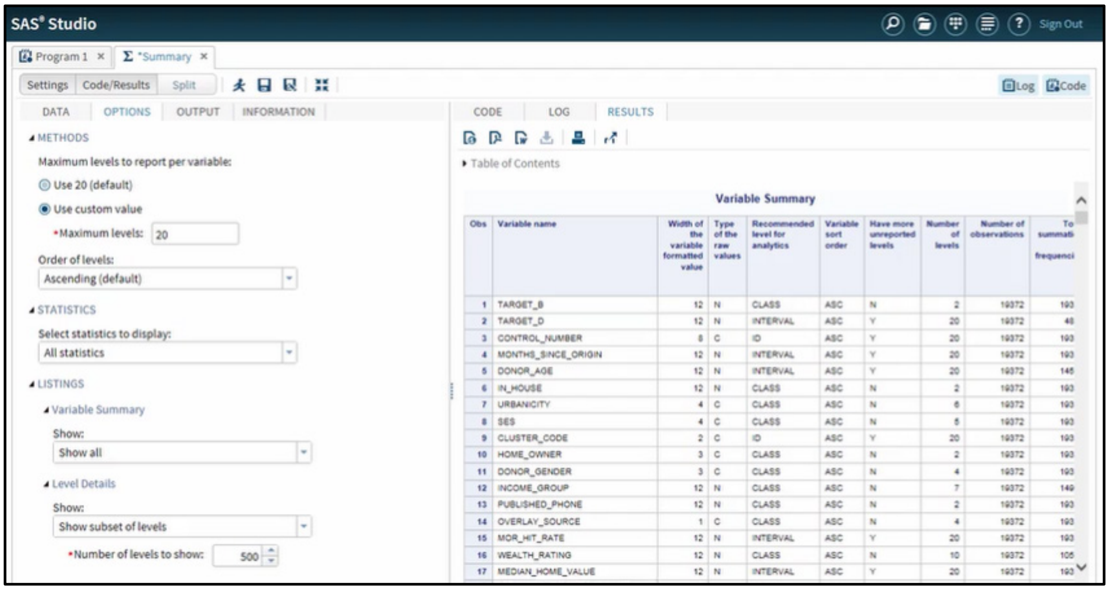
```
ods noproctitle;

proc cardinality data=MYCAS.DONOR_RAW_DATA maxlevels=
                outcard=MYCAS.varSummaryTemp out=MYCA
run;

proc print data=MYCAS.varSummaryTemp label;
        var _varname_ _fmtwidth_ _type_ _rlevel_ _mo
                _max_ _mean_ _stddev_;
        title "Variable Summary";
run;

proc print data=MYCAS.levelDetailTemp (obs=20) label;
```

Line 1, Column 1

Messages: 35

Then, we will set the number of report observations to 500 in the Number of levels to show field. To run this task, click on the Running Person at the top. To view your results in a larger window, you can click the Maximize View option. The Variable Summary Report is shown in Figure 3.3. Scroll down to view the Level Details report.

Figure 3.3: Variable Summary Report



METHODS

Maximum levels to report per variable:

Use 20 (default) [✓ Use custom value]

*Maximum levels: 20

Order of levels: Ascending (default)

STATISTICS

Select statistics to display: All statistics

LISTINGS

Variable Summary

Show: Show all

Level Details

Show: Show subset of levels

*Number of levels to show: 500

Variable Summary

Obs	Variable name	Width of the variable formatted value	Type of the raw values	Recommended level for analytics	Variable sort order	Have more unreported levels	Number of levels	Number of observations	To summati frequenci
1	TARGET_B	12	N	CLASS	ASC	N	2	19372	193
2	TARGET_D	12	N	INTERVAL	ASC	Y	20	19372	48
3	CONTROL_NUMBER	8	C	ID	ASC	Y	20	19372	193
4	MONTHS_SINCE_ORIGIN	12	N	INTERVAL	ASC	Y	20	19372	193
5	DONOR_AGE	12	N	INTERVAL	ASC	Y	20	19372	145
6	IN_HOUSE	12	N	CLASS	ASC	N	2	19372	193
7	URBANICITY	4	C	CLASS	ASC	N	6	19372	193
8	SES	4	C	CLASS	ASC	N	5	19372	193
9	CLUSTER_CODE	2	C	ID	ASC	Y	20	19372	193
10	HOME_OWNER	3	C	CLASS	ASC	N	2	19372	193
11	DONOR_GENDER	3	C	CLASS	ASC	N	4	19372	193
12	INCOME_GROUP	12	N	CLASS	ASC	N	7	19372	149
13	PUBLISHED_PHONE	12	N	CLASS	ASC	N	2	19372	193
14	OVERLAY_SOURCE	1	C	CLASS	ASC	N	4	19372	193
15	MOR_HIT_RATE	12	N	INTERVAL	ASC	Y	20	19372	193
16	WEALTH_RATING	12	N	CLASS	ASC	N	10	19372	105
17	MEDIAN_HOME_VALUE	12	N	INTERVAL	ASC	Y	20	19372	193

A level recommendation is provided for each variable. This recommendation is useful as a starting point. For example, the recommended level for the variable MONTHS_SINCE_ORIGIN is Interval. On the Levels Detail Report, the column TARGET_B has level 0 and 1, and level 1 is 25% of the total.

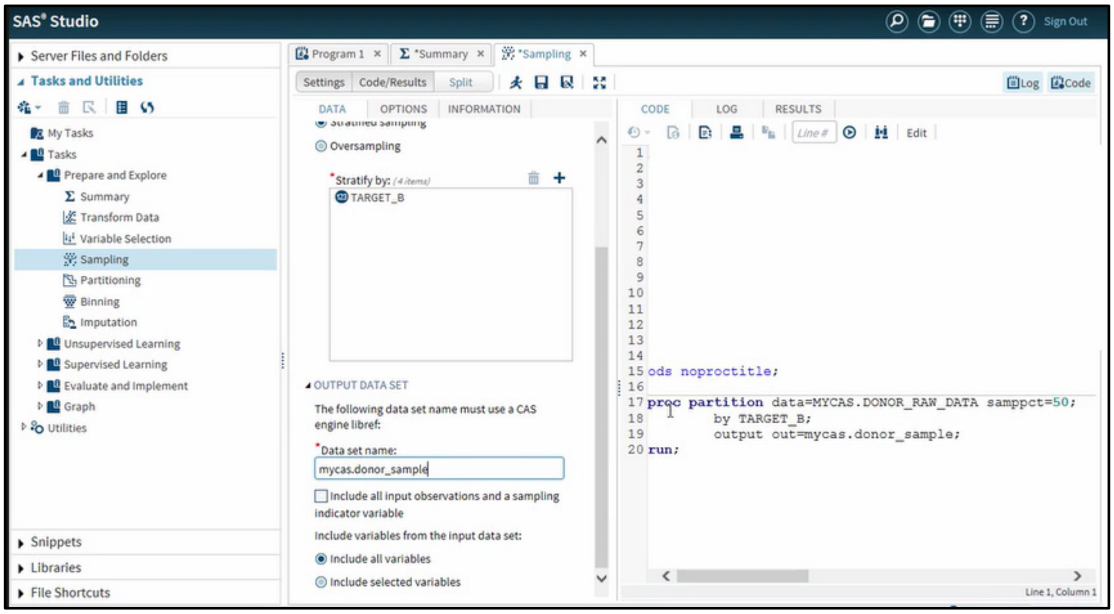
If you want to manually edit the code, click on the code tab to view the code. Then click edit to edit the code.

Sampling

Now, we will learn how to sample your data. To accomplish this, we will use the Sampling task. Double click on Sampling from the Prepare and Explore folder in the left menu.

Select your data on the data tab. In this example, we want to perform stratified sampling on TARGET_B, so under Sampling Method, we will select Stratified Sampling and then add the stratified variable, TARGET_B. Under Output Data Set, we will enter the output table name. Notice that the code PROC PARTITION is being created as you select your options, as shown in Figure 3.4.

Figure 3.4: Sampling Data Tab



Oversampling

*Stratify by: (4 items)
TARGET_B

OUTPUT DATA SET

The following data set name must use a CAS engine libref:

*Data set name: mycas.donor_sample

☐ Include all input observations and a sampling indicator variable

Include variables from the input data set:

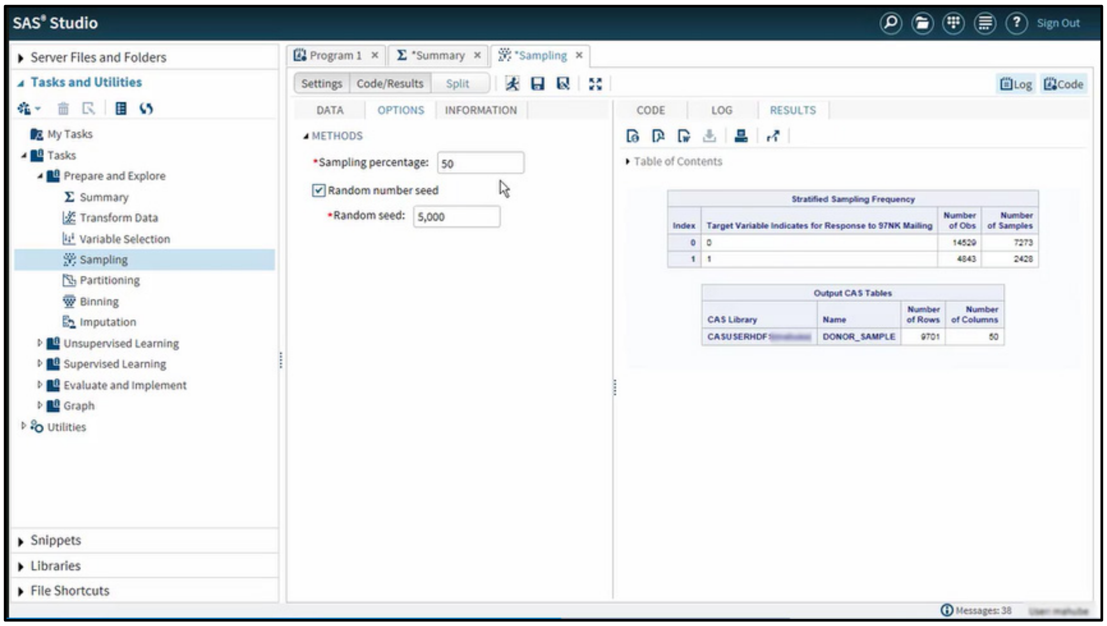
● Include all variables

○ Include selected variables

```
ods noproctitle;

proc partition data=MYCAS.DONOR_RAW_DATA samppct=50;
        by TARGET_B;
        output out=mycas.donor_sample;
run;
```

On the Options Tab, we will use the default 50% sampling rate and arbitrarily enter 5,000 for the random seed. After you run the data, notice that in the Frequency report shown in Figure 3.5, 50% of the data has been sampled and that TARGET_B has the same distribution in the original data, which is a 25/75 split.

Figure 3.5: Frequency Report



METHODS

*Sampling percentage: 50

☑ Random number seed

*Random seed: 5,000

Stratified Sampling Frequency

Index	Target Variable Indicates for Response to 97NK Mailing	Number of Obs	Number of Samples
0	0	14529	7273
1	1	4843	2428

Output CAS Tables

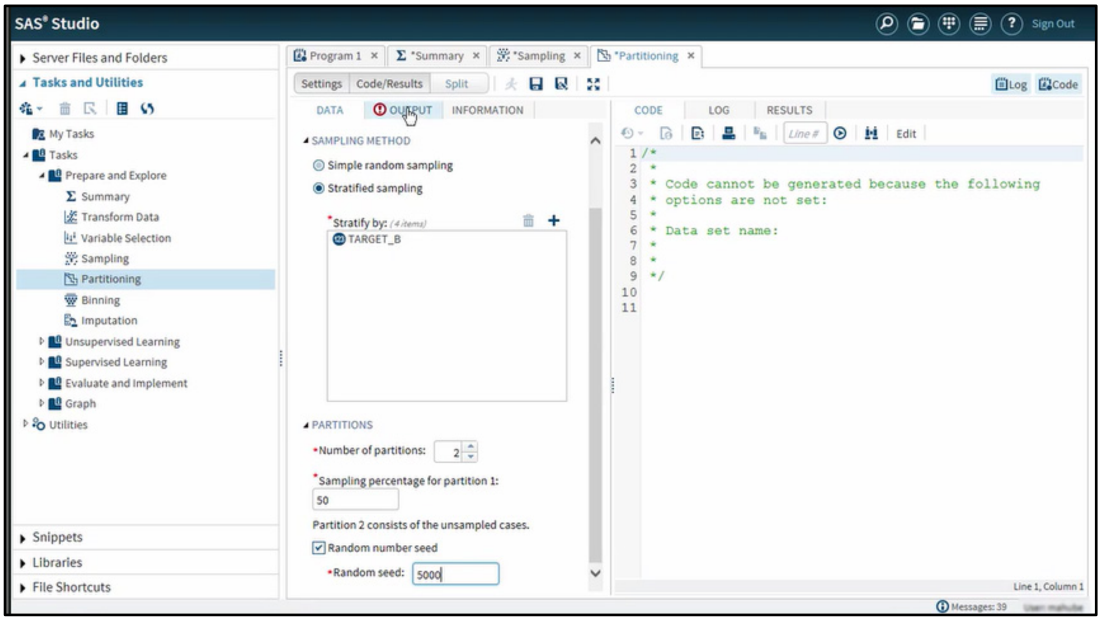
CAS Library	Name	Number of Rows	Number of Columns
CASUSERHDF[redacted]	DONOR_SAMPLE	9701	50

You can manually edit the program by clicking on the Code tab above the Frequency Report, then clicking Edit.

Partitioning

Lastly, we will learn how to partition data. To accomplish this, we will use the Partitioning task. Double click on Partitioning from the Prepare and Explore folder in the left menu. Select your data in the Data tab. In this example, we will be performing stratified sampling on TARGET_B, so we will select Stratified Sampling under the Sampling Method and then select the appropriate column. In the Partitions section, we will enter 2 in the Number of Partitions field, and a 50% split in the Sampling percentage field. We will arbitrarily enter 5000 for the random seed, as shown in Figure 3.6.

Figure 3.6: Partitioning Data Tab



SAMPLING METHOD

Simple random sampling

Stratified sampling [✓]

*Stratify by: (4 items)
TARGET_B

PARTITIONS

*Number of partitions: 2

*Sampling percentage for partition 1: 50

Partition 2 consists of the unsampled cases.

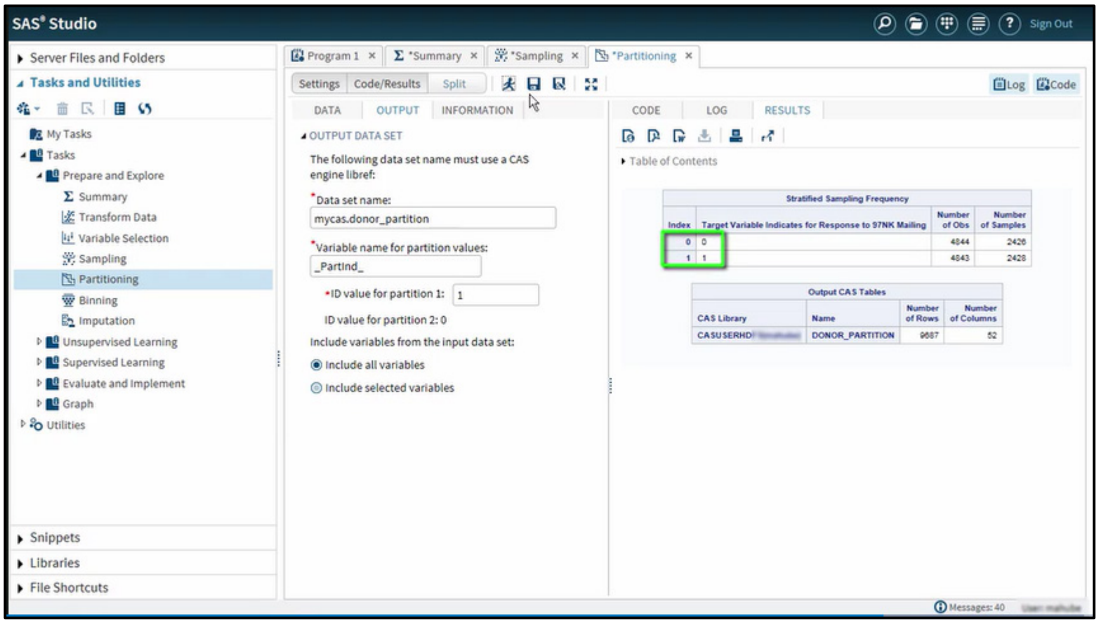
[✓] Random number seed

*Random seed: 5000

```
/*
 *
 * Code cannot be generated because the following
 * options are not set:
 *
 * Data set name:
 *
 *
 */
```

Under the Options tab, enter your output table name. Notice that the default name for the partition values is given to you as _Partind_. After you run the task, notice that the column _PARTIND_ has the value 0 or 1 as shown in Figure 3.7. 1 is for the training data, and 0 is for the validation data. Also notice the 50/50 split for the partition data and the 50/50 split for TARGET_B, which is what we specified.

Figure 3.7: Stratified Sampling Frequency Report



OUTPUT DATA SET

The following data set name must use a CAS engine libref:

*Data set name: mycas.donor_partition

*Variable name for partition values: _PartInd_

*ID value for partition 1: 1

ID value for partition 2: 0

Include variables from the input data set:

Include all variables [✓]

Include selected variables

Stratified Sampling Frequency

Index	Target Variable Indicates for Response to 97NK Mailing	Number of Obs	Number of Samples
0	0	4844	2426
1	1	4843	2428

Output CAS Tables

CAS Library	Name	Number of Rows	Number of Columns
CASUSERHD[redacted]	DONOR_PARTITION	9687	52

You can manually edit the program by clicking on the Code tab above the Report, then clicking Edit.

Data Transformation

In this section, you will learn how to use SAS Viya to perform data transformation tasks such as variable imputation and variable bending. We will continue to use the same data source used in the previous examples, which contains a list of donors to an organization. This source has a target variable, TARGET_B, which is a binary variable with a value of 1 for a person who donated during a mailing.

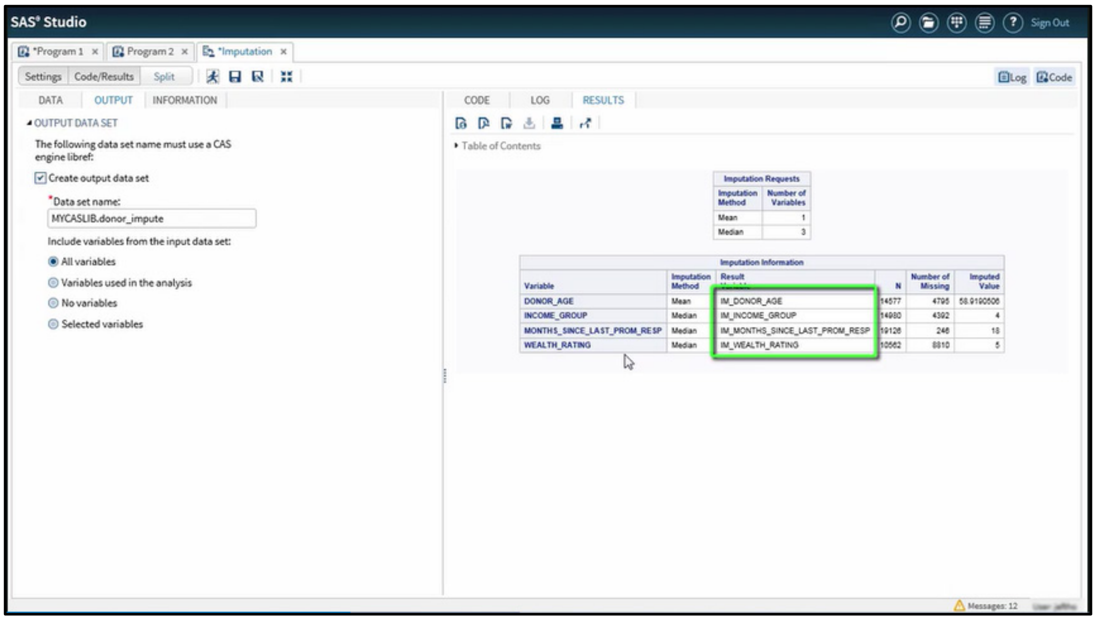
Imputation

First, we will learn how to impute missing values. To accomplish this, we will use the Imputation task. Double click on Imputation from the Prepare and Explore folder in the left menu.

Then, select your data in the Data tab that opens in the Summary window. Next, you need to select variables for mean imputation. In this example, we will select DONOR_AGE for the mean imputation by clicking on the + sign in the upper right corner of the first box in the Roles group. In the next box, use the + to sign select variables for median imputation. In this example, we are selecting INCOME_GROUP, WEALTH_RATING, and MONTHS_SINCE_LAST_PROM_RESP, each of which contain missing values.

On the output tab, choose a name for the data set in the Data set name box. In this example, we will select All variables to include in the input data set. The results after clicking Run are shown in Figure 3.8.

Figure 3.8: Imputation Results



OUTPUT DATA SET

The following data set name must use a CAS engine libref:

☑ Create output data set

*Data set name: MYCASLIB.donor_impute

Include variables from the input data set:

[✓ All variables]
Variables used in the analysis
No variables
Selected variables

Imputation Requests	
Imputation Method	Number of Variables
Mean	1
Median	3

Imputation Information					
Variable	Imputation Method	Result Variable	N	Number of Missing	Imputed Value
DONOR_AGE	Mean	IM_DONOR_AGE	14577	4795	58.9190506
INCOME_GROUP	Median	IM_INCOME_GROUP	14980	4392	4
MONTHS_SINCE_LAST_PROM_RESP	Median	IM_MONTHS_SINCE_LAST_PROM_RESP	19126	246	18
WEALTH_RATING	Median	IM_WEALTH_RATING	10562	8810	5

Notice that the generated variables are named by prepending IM_ to the original variable names. You can manually edit the program by clicking on the Code tab above the Report, then clicking Edit.

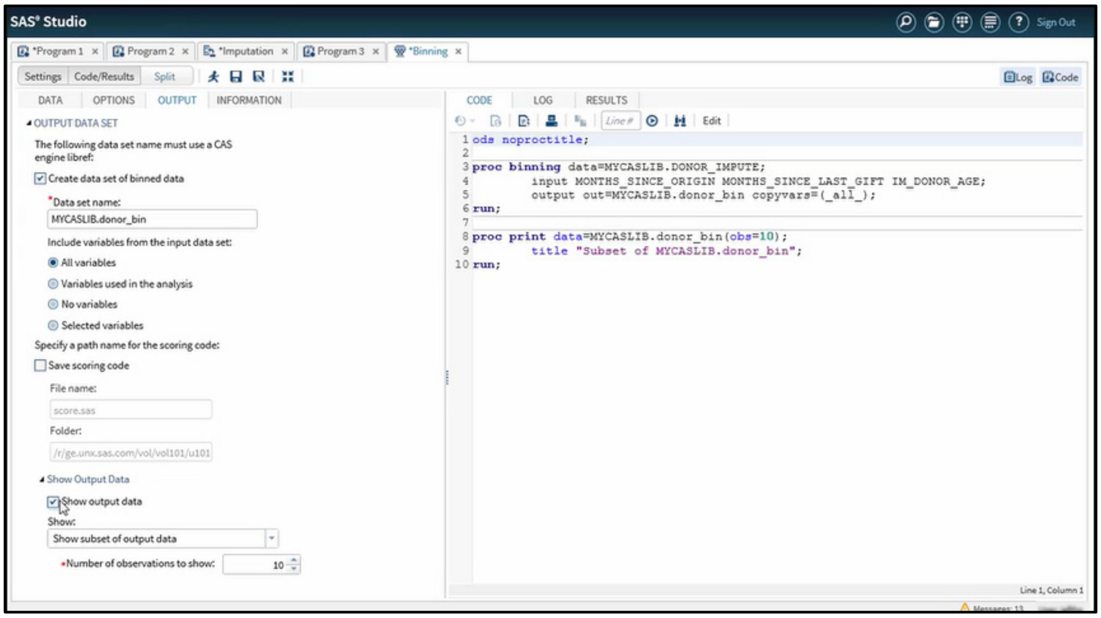
Binning

Now we will learn how to bin interval variables. To accomplish this, we will use the Binning task. Double click on Binning from the Prepare and Explore folder in the left menu.

Select your data in the Data tab that opens in the Summary window. Next, you need to select the variables that you want to bin in the Roles section. Click on the + sign to select variables. In this example, we are selecting three variables: MONTHS_SINCE_ORIGIN, MONTHS_SINCE_LAST_GIFT, and IM_DONOR_AGE.

Click the Options tab. The default method is Bucket Binning to bin each variable into 16 bins. Next, click the Output tab. If you want to create a data set from your binned data, click the check box to turn on that feature. Then specify the name of your new data set. In this example, we will select the options to include all variables and also show output, as shown in Figure 3.9.

Figure 3.9: Binning Output Tab



OUTPUT DATA SET

The following data set name must use a CAS engine libref:

[✓] Create data set of binned data

*Data set name: MYCASLIB.donor_bin

Include variables from the input data set:

- [✓ All variables]
- Variables used in the analysis
- No variables
- Selected variables

Specify a path name for the scoring code:

[ ] Save scoring code

File name: score.sas

Folder: /r/ge.unx.sas.com/vol/vol101/u101

Show Output Data

[✓] Show output data

Show: Show subset of output data

*Number of observations to show: 10

```
ods noproctitle;

proc binning data=MYCASLIB.DONOR_IMPUTE;
        input MONTHS_SINCE_ORIGIN MONTHS_SINCE_LAST_GIFT IM_DONOR_AGE;
        output out=MYCASLIB.donor_bin copyvars=(_all_);
run;

proc print data=MYCASLIB.donor_bin(obs=10);
        title "Subset of MYCASLIB.donor_bin";
run;
```

After you run the task, you will se the Bin Details report, which gives information about each of the 16 bins for each variable. Below that, you can scroll down to view the output table. Notice that the 3 binned variables that we selected earlier have bin_ prepended to their names and that they contain the bin ID values.

You can manually edit the program by clicking on the Code tab above the Report, then clicking Edit.

Variable Selection Techniques

In this section, you will learn how to perform unsupervised variable reduction using PROC VARREDUCE in SAS Viya. If you have used PROC VARCLUS in SAS 9 to perform variable reduction, then keep reading to see how the new procedure compares.

A common reason for performing variable reduction is that the size of your sample cannot support the number of effects available for modeling. Eliminating redundancy in your effects before modeling is the goal. Principal Component Analysis is an option, but it is a dimension reduction technique, not a variable reduction technique. If you need to be able to interpret the model inputs, principal components are not a good choice. Another technique called variable clustering is available in SAS 9. It is performed by the VARCLUS procedure.

Unsupervised Variable Reduction Code in SAS Viya versus SAS 9

PROC VARCLUS in SAS 9 finds disjoint clusters of variables with high correlations. When the clusters are found, you have the option of using either the principal component score or a representative variable from each cluster in subsequent analysis.

In SAS Viya, you can use the VARREDUCE procedure, one of the SAS Visual Data Mining and Machine Learning statistics procedures. PROC VARREDUCE uses an entirely different algorithm to perform unsupervised variable reduction. It can perform supervised variable reduction as well. Let’s look at an example of unsupervised variable reduction that shows how you can achieve the same goals that you can with PROC VARCLUS.

SAS 9 Code

For this example, we will use the getStarted data set, which is available from the SAS documentation. The variables created are the generically named X1 through X10 for the numeric inputs, Y for the binary target, and C for the categorical input variable. The binary target is needed only when you perform supervised data reduction.

Program 3.1 shows an example of code for variable reduction using PROC VARCLUS in SAS 9.

Program 3.1: SAS 9 Code

```
proc varclus data=work.getStarted maxeigen=1.0 short maxclusters=5; ❶
   var x1-x10;
run;
```

❶ This code is fairly simple. We are specifying two stopping criteria. One is the MAXEIGEN criterion, which stops the algorithm when all clusters at a step have second eigenvalues below the specified value. The other is the MAXCLUSTERS criterion, which stops the algorithm when the specified number of clusters is reached.

The output from running this code shows a summary of the number of clusters along with the members of each cluster. The One minus R-squared table helps you select the most representative variable from each cluster.

SAS Viya Code

Now let's look at the code for PROC VARREDUCE in Program 3.2.

Program 3.2: SAS Viya Code

```
proc varreduce data=mycaslib.getStarted technique=VarianceAnalysis; ❶
   class C; ❷
   reduce unsupervised C x1-x10 / maxeffects=5 varexp=0.9; ❸
run;
```

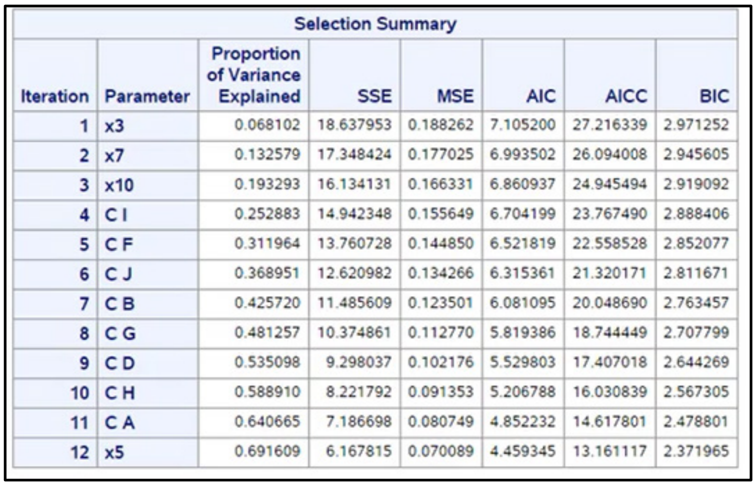
❶ We specify that we would like to use variance analysis to select variables. Using this method, PROC VARREDUCE continues to add variables until a specified proportion of the total variance is reached.

❷ The CLASS statement creates dummy variables for the variable C. Unlike PROC VARCLUS, PROC VARREDUCE accepts both categorical and interval inputs.

❸ Next is the REDUCE statement, which specifies that we are performing unsupervised variable reduction. Two stopping criteria are also mentioned here. The MAXEFFECTS option stops the algorithm when the specified number of effects are added. The VAREXP option stops the algorithm when the proportion of total variance is achieved.

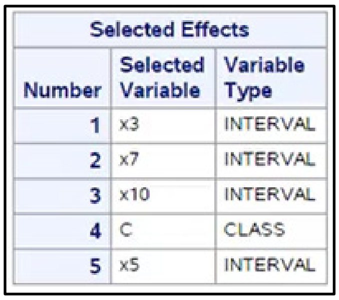
There isn't much output from the procedure. The selection summary table in Output 3.2 shows the progression of proportion of variance explained as each parameter is added. Notice in that dummy variables for C were added individually. The proportion of variance never reached 0.9 because the algorithm already added the maximum of five effects. Other summary fit statistics are reported in the selection summary table. In addition to accepting categorical inputs, PROC VARREDUCE can also assess interaction, polynomial, and nested effects.

Output 3.2: Results of Program 3.2



Selection Summary

Iteration	Parameter	Proportion of Variance Explained	SSE	MSE	AIC	AICC	BIC
1	x3	0.068102	18.637953	0.188262	7.105200	27.216339	2.971252
2	x7	0.132579	17.348424	0.177025	6.993502	26.094008	2.945605
3	x10	0.193293	16.134131	0.166331	6.860937	24.945494	2.919092
4	C I	0.252883	14.942348	0.155649	6.704199	23.767490	2.888406
5	C F	0.311964	13.760728	0.144850	6.521819	22.558528	2.852077
6	C J	0.368951	12.620982	0.134266	6.315361	21.320171	2.811671
7	C B	0.425720	11.485609	0.123501	6.081095	20.048690	2.763457
8	C G	0.481257	10.374861	0.112770	5.819386	18.744449	2.707799
9	C D	0.535098	9.298037	0.102176	5.529803	17.407018	2.644269
10	C H	0.588910	8.221792	0.091353	5.206788	16.030839	2.567305
11	C A	0.640665	7.186698	0.080749	4.852232	14.617801	2.478801
12	x5	0.691609	6.167815	0.070089	4.459345	13.161117	2.371965



Selected Effects		
Number	Selected Variable	Variable Type
1	x3	INTERVAL
2	x7	INTERVAL
3	x10	INTERVAL
4	C	CLASS
5	x5	INTERVAL

Unsupervised Variable Reduction Example

Let's look at another, more complex example of performing unsupervised feature reduction with SAS Viya.

Program 3.3: Unsupervised Variable Reduction

```
/*View the Data*/ ❶
proc print data=mycaslib.featuredemo (obs=10);
run;

/*Investigate the Categorical Variable*/ ❷
proc cardinality data=mycaslib.feature demo
        outcard=mycaslib.card outdetails=mycaslib.details;
   var cat_features;
run;

proc print data=mycaslib.card;
run;

proc print data=mycaslib.details;
run;


/*Unsupervised Variable Reduction*/ ❸
proc varreduce data=mycaslib.featuredemo technique=Varianceanalysis; ❹
   class cat_feature; ❺
   reduce unsupervised cat_feature feature_1 - feature_10 /
          maxeffect=5 varexp=0.95; ❻
run;
```

❶ The first statement of Program 3.3 lets us view the data. The data we have in this example consists of a binary label, ten numeric features, and one categorical feature, as shown in Output 3.3a.

❷ The next statement in the code lets us view the categorical feature in more detail. Our categorical feature has nine levels, as seen in Output 3.3b.

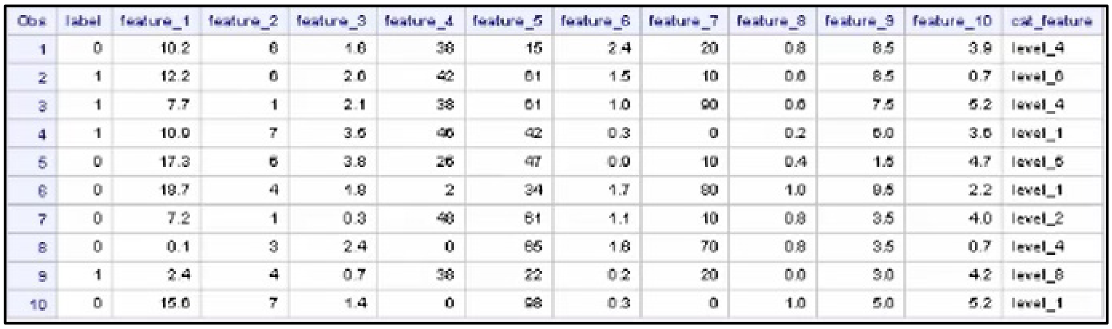
❸ We can use the categorical feature in the VARREDUCE procedure without needing to explicitly create a separate encoding for it as a prior step. The VARREDUCE procedure automatically creates a one hot encoding of this feature during processing.

❹ After invoking the VARREDUCE procedure, we name the input data set and specify the technique that we want to employ for dimension reduction. In this instance, we used variance analysis.

❺ The CLASS statement indicates the features for which special encoding is to be conducted before the application of the reduction technique.

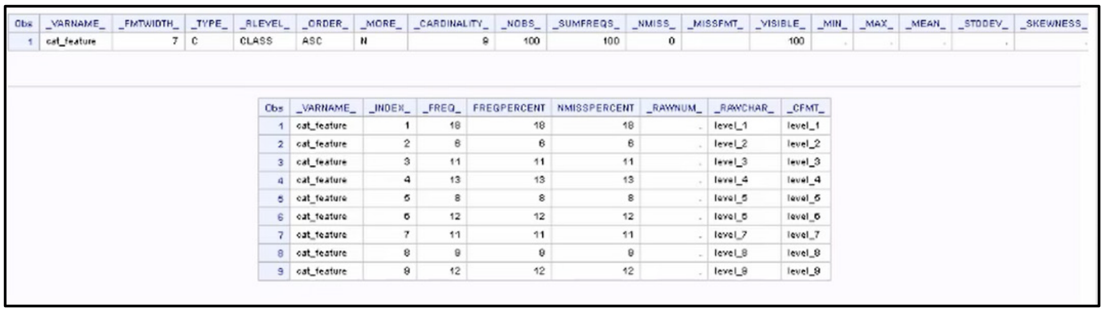
❻ The REDUCE statement specifies the type of selection to conduct – unsupervised in this case – and the features to be considered in the feature selection process. The REDUCE options, which are found after the forward slash in the statement, control the number of variables to be selected. Here we indicate a desire for either a maximum of 5 features or the number of variables that explain at least 95% of the variance in the data, whichever comes first. The output for this procedure is shown in Output 3.3c.

Output 3.3a: Data



Obs	label	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	cat_feature
1	0	10.2	6	1.6	38	15	2.4	20	0.8	8.5	3.9	level_4
2	1	12.2	6	2.6	42	61	1.5	10	0.0	8.5	0.7	level_6
3	1	7.7	1	2.1	38	61	1.0	90	0.6	7.5	5.2	level_4
4	1	10.9	7	3.6	46	42	0.3	0	0.2	6.0	3.6	level_1
5	0	17.3	6	3.8	26	47	0.0	10	0.4	1.5	4.7	level_5
6	0	18.7	4	1.8	2	34	1.7	80	1.0	8.5	2.2	level_1
7	0	7.2	1	0.3	48	61	1.1	10	0.8	3.5	4.0	level_2
8	0	0.1	3	2.4	0	65	1.8	70	0.8	3.5	0.7	level_4
9	1	2.4	4	0.7	38	22	0.2	20	0.0	3.0	4.2	level_8
10	0	15.6	7	1.4	0	98	0.3	0	1.0	5.0	5.2	level_1

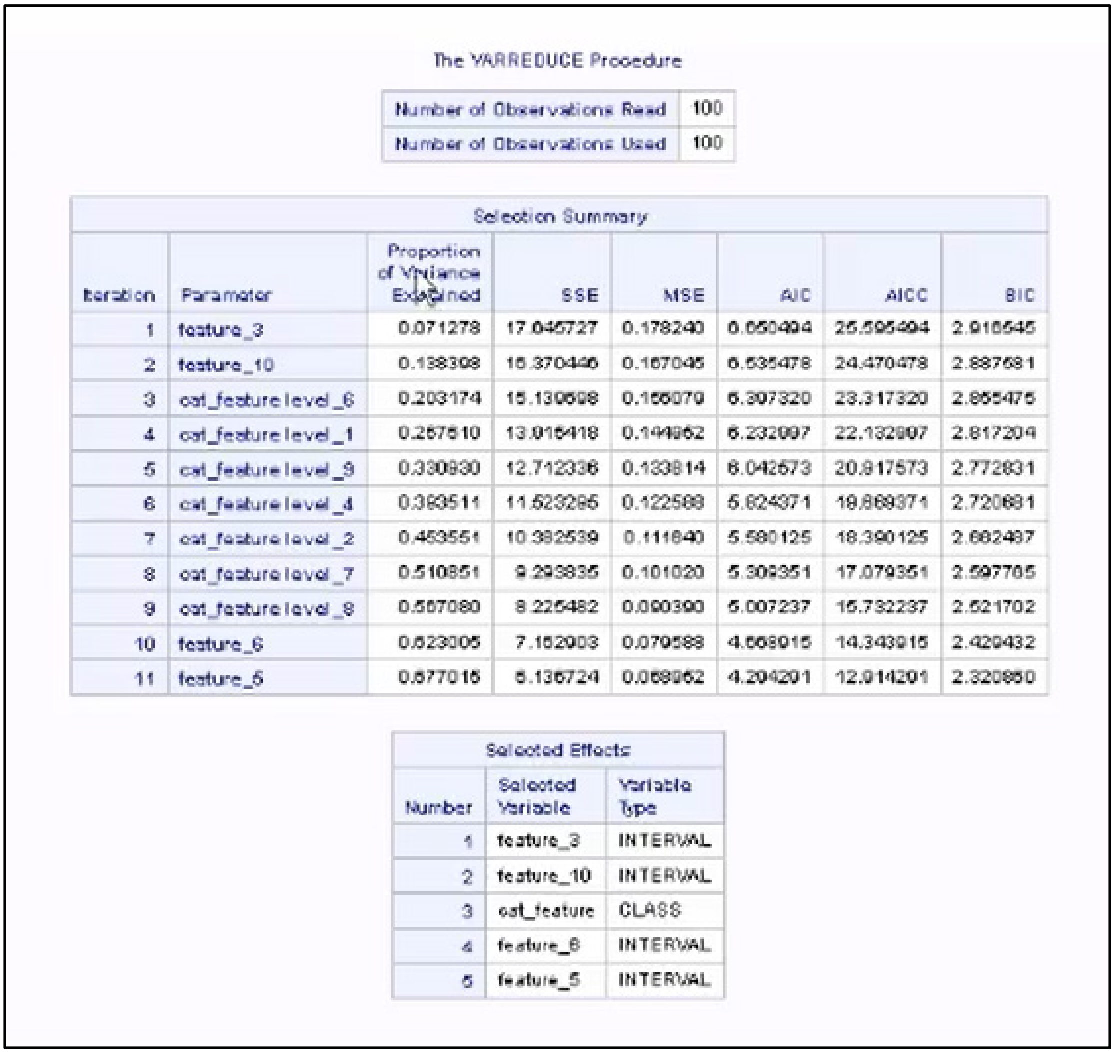
Output 3.3b: Categorical Feature



Obs	_VARNAME_	_FMTWIDTH_	_TYPE_	_RLEVEL_	_ORDER_	_MORE_	_CARDINALITY_	_NOBS_	_SUMFREQS_	_NMISS_	_MISSFMT_	_VISIBLE_	_MIN_	_MAX_	_MEAN_	_STDDEV_	_SKEWNESS_
1	cat_feature	7	C	CLASS	ASC	N	9	100	100	0		100	.	.	.	.	.

Obs	_VARNAME_	_INDEX_	_FREQ_	FREQPERCENT	NMISSPERCENT	_RAWNUM_	_RAWCHAR_	_CFMT_
1	cat_feature	1	18	18	18	.	level_1	level_1
2	cat_feature	2	6	6	6	.	level_2	level_2
3	cat_feature	3	11	11	11	.	level_3	level_3
4	cat_feature	4	13	13	13	.	level_4	level_4
5	cat_feature	5	8	8	8	.	level_5	level_5
6	cat_feature	6	12	12	12	.	level_6	level_6
7	cat_feature	7	11	11	11	.	level_7	level_7
8	cat_feature	8	9	9	9	.	level_8	level_8
9	cat_feature	9	12	12	12	.	level_9	level_9

Output 3.3c: VARREDUCE Procedure – Unsupervised



The VARREDUCE Procedure

Number of Observations Read	100
Number of Observations Used	100

Selection Summary

Iteration	Parameter	Proportion of Variance Explained	SSE	MSE	AIC	AICC	BIC
1	feature_3	0.071278	17.045727	0.178240	6.650494	25.595494	2.916545
2	feature_10	0.138398	16.370446	0.167045	6.535478	24.470478	2.887681
3	cat_feature level _6	0.203174	15.139698	0.156079	6.397320	23.317320	2.855475
4	cat_feature level _1	0.267610	13.916418	0.144962	6.232997	22.132997	2.817204
5	cat_feature level _9	0.330830	12.712336	0.133814	6.042573	20.917573	2.772831
6	cat_feature level _4	0.393511	11.523295	0.122588	5.824371	19.669371	2.720681
7	cat_feature level _2	0.453551	10.382539	0.111640	5.580125	18.390125	2.662487
8	cat_feature level _7	0.510851	9.293835	0.101020	5.309351	17.079351	2.597765
9	cat_feature level _8	0.567080	8.225482	0.090390	5.007237	15.732237	2.521702
10	feature_6	0.623005	7.162903	0.079588	4.668915	14.343915	2.429432
11	feature_5	0.677016	6.136724	0.068962	4.294201	12.914201	2.320860

Selected Effects

Number	Selected Variable	Variable Type
1	feature_3	INTERVAL
2	feature_10	INTERVAL
3	cat_feature	CLASS
4	feature_6	INTERVAL
5	feature_5	INTERVAL

A summary of the selection process is presented in Output 3.3c. We can see that five variables have been selected.

Supervised Feature Selection Example

Now let's view an example of supervised feature selection in Program 3.4.

Program 3.4: Supervised Variable Reduction

```
/*Unsupervised Variable Reduction*/
proc varreduce data=mycaslib.featuredemo technique=discriminantanalysis; ❶
   class cat_feature label; ❷
   reduce supervised cat_feature feature_1 - feature_10 / maxeffect=5; ❸
   ods output selectionsummary = summary; ❹
run;
```

❶ We invoke the PROC VARREDUCE procedure, as we did in the unsupervised case in Program 3.3. But here we switch the technique employed to discriminant analysis.

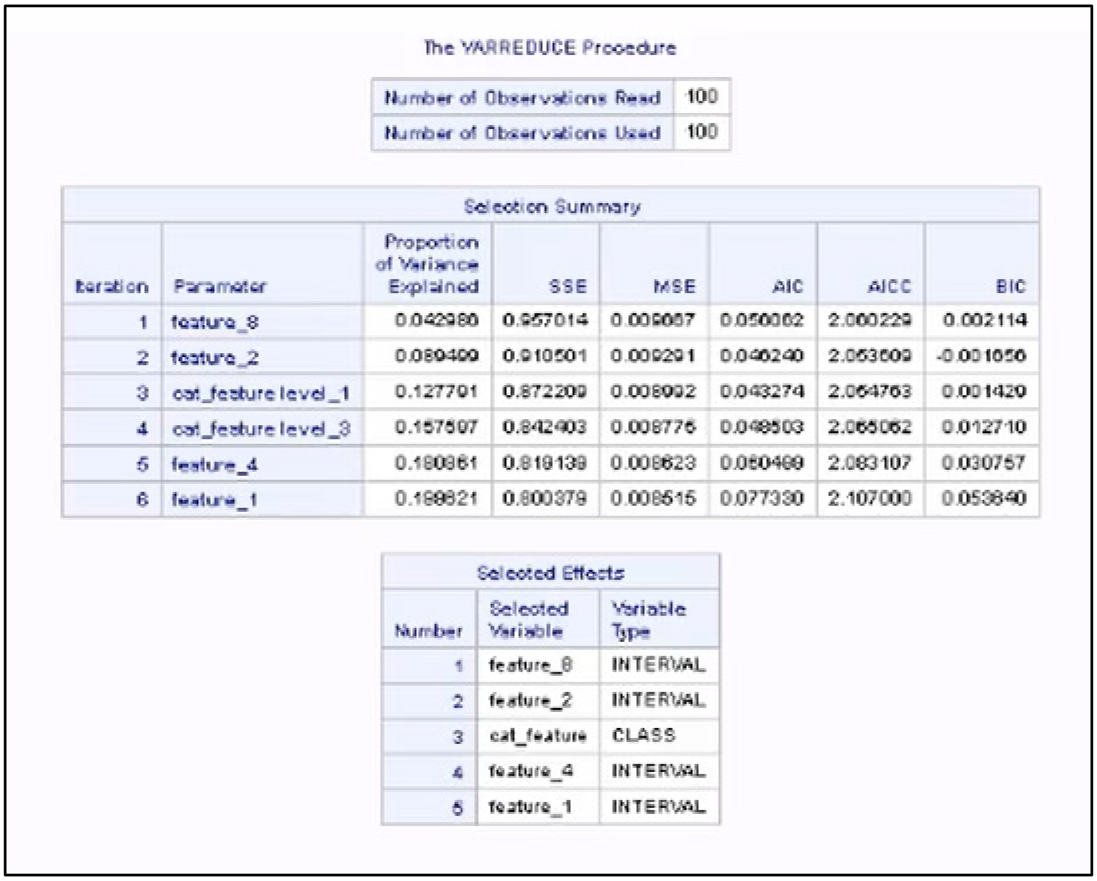
❷ We add our binary label to the CLASS statement so that encoding is conducted.

❸ In the REDUCE statement, we specify that the type of selection to use is now supervised and the binary label is the outcome that we are interested in modeling.

❹ We add an ODS OUTPUT statement to save summary statistics in an output data set.

Let's run Program 3.4 and view the output in Output 3.4.

Output 3.4: The VARREDUCE Procedure – Supervised



The VARREDUCE Procedure

Number of Observations Read	100
Number of Observations Used	100

Selection Summary

Iteration	Parameter	Proportion of Variance Explained	SSE	MSE	AIC	AICC	BIC
1	feature_8	0.042986	0.957014	0.009067	0.050062	2.060229	0.002114
2	feature_2	0.089499	0.910501	0.009291	0.046240	2.053609	-0.001656
3	cat_feature level_1	0.127791	0.872209	0.008992	0.043274	2.054763	0.001429
4	cat_feature level_3	0.157597	0.842403	0.008776	0.048503	2.065062	0.012710
5	feature_4	0.180861	0.819139	0.008623	0.060499	2.083107	0.030757
6	feature_1	0.199621	0.800379	0.008515	0.077330	2.107000	0.053840

Selected Effects

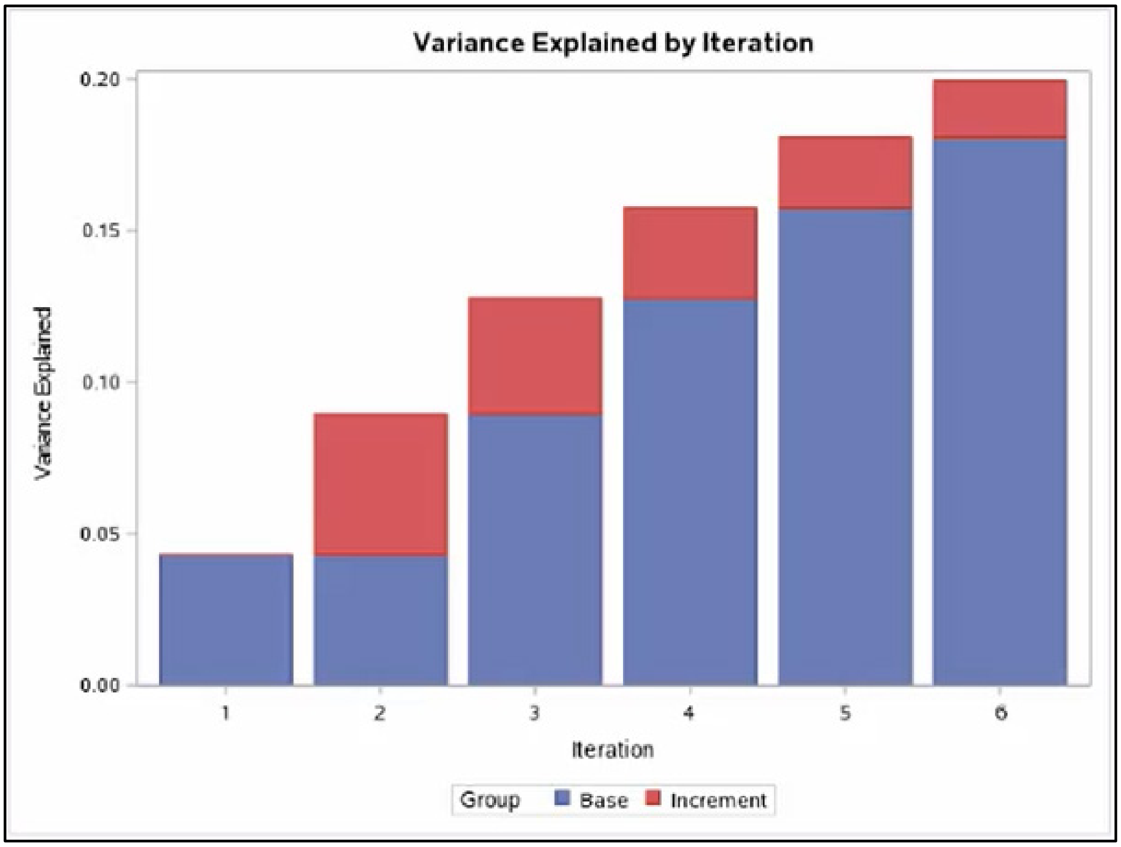
Number	Selected Variable	Variable Type
1	feature_8	INTERVAL
2	feature_2	INTERVAL
3	cat_feature	CLASS
4	feature_4	INTERVAL
5	feature_1	INTERVAL

A summary of the selection process is presented in Output 3.4 and we can see that 5 variables have been selected. We can make use of the output summary statistics by creating a graphic that reveals the amount of variation explained by the iteration of the selection process. The code to create this graphic is given in Program 3.5 and the graphic is shown in Output 3.5.

Program 3.5: Code to Create Graphic

```
data out_iter (keep=Iteration VarExp Vase Increment Variable);
   set summary;
   Increment=dif(VarExp);
   if Increment='.' Then
          Increment=0;
   Base=VarExp - Increment;
run;

proc transpose data=out_iter out=out_iter_trans;
   by Iteration VarExp Variable;
run;

proc sort data=out_iter_trans;
   label _NAME_='Group';
   by _NAME_;
run;

title "Variance Explained by Iteration";

proc sgplot data=out_iter_trans;
   yaxis label="Variance Explained";
   vbar Iteration / response=COL1 group=_NAME_;
run;
```

Output 3.5: Results of Program 3.5



Unsupervised Learning

In this section, you will learn how to use SAS Viya to perform unsupervised learning tasks, including Clustering and Principal Component Analysis (PCA).

Here are the procedures included in the SAS Studio tasks that will be discussed in this section:

Name	SAS Studio Task	SAS Viya Procedure	SAS 9 Procedure
K-means Clustering	Clustering	PROC KCLUS	PROC FASTCLUS
Principal Component Analysis	Principal Component Analysis	PROC PCA	PROC PRINCOMP

K-Means Clustering

In this section, you will learn about k-means clustering, which falls under the umbrella of unsupervised learning. The goal of clustering is to group observations into categories such that within-cluster or group variability is minimized and between-cluster variability is maximized. In the end, every observation belongs to exactly one cluster.

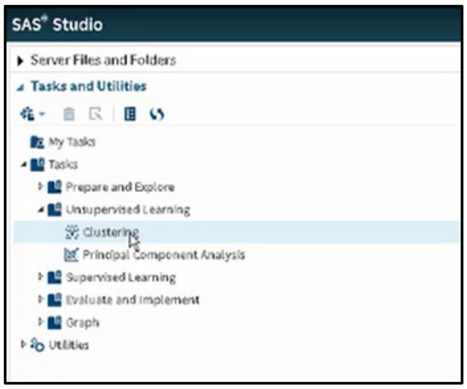
Let's look at this technique more closely with an example. For this example, we will use the cars data set in the SASHELP library. It contains information about cars, including their makes, models, manufacturers, and other characteristics. The first step is to load the cars data set from the SASHELP library into the MYCASLIB library. Note that dummy variables are created for the nominal inputs, Origin and DriveTrain.

To do k-means clustering, you can use a SAS Studio task or run code. We will look at both methods, starting with the task.

Clustering Task in SAS Studio

Click the Task and Utilities tab on the left in SAS Studio. Expand Tasks and Unsupervised Learning, as shown in Figure 3.10. Double click on Clustering to open the Clustering window.

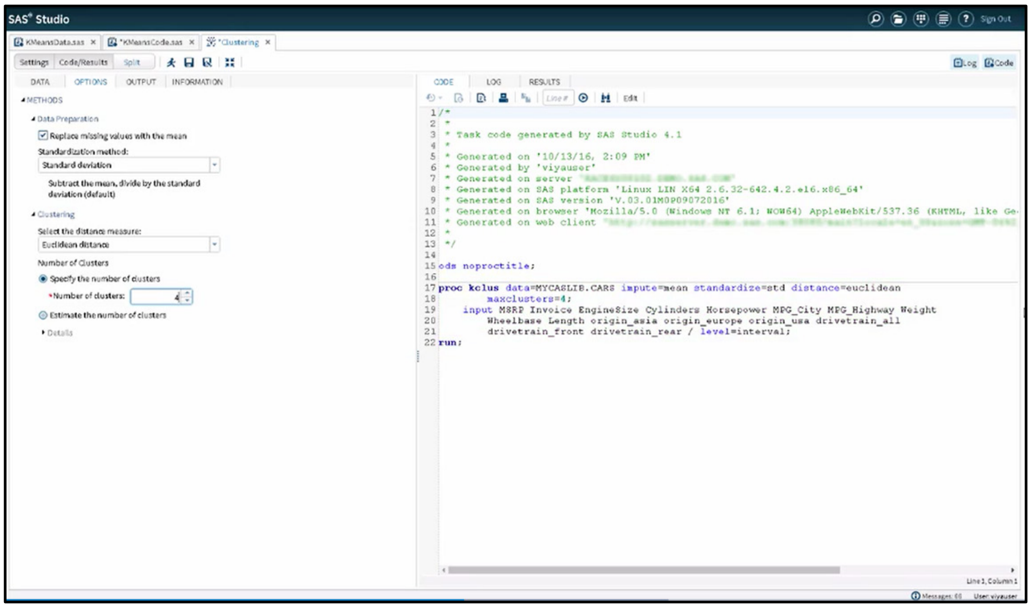
Figure 3.10: Unsupervised Learning Tasks



Select the cars data set in the MYCASLIB library, then select all available interval inputs in the data by clicking on the + sign next to the interval inputs window and selecting all variables.

Go to the options tab and perform these actions as shown in Figure 3.11: select the Replace Missing Values with the Mean check box, select Standard Deviation from the Standardization method menu, specify 4 as the number of clusters. In SAS Studio, the code window on the right generates the necessary SAS code to perform the k-means clustering. Run the task by clicking the running person.

Figure 3.11: Clustering Options tab



METHODS

Data Preparation

☑ Replace missing values with the mean

Standardization method: Standard deviation

Subtract the mean, divide by the standard deviation (default)

Clustering

Select the distance measure: Euclidean distance

Number of Clusters

● Specify the number of clusters

*Number of clusters: 4

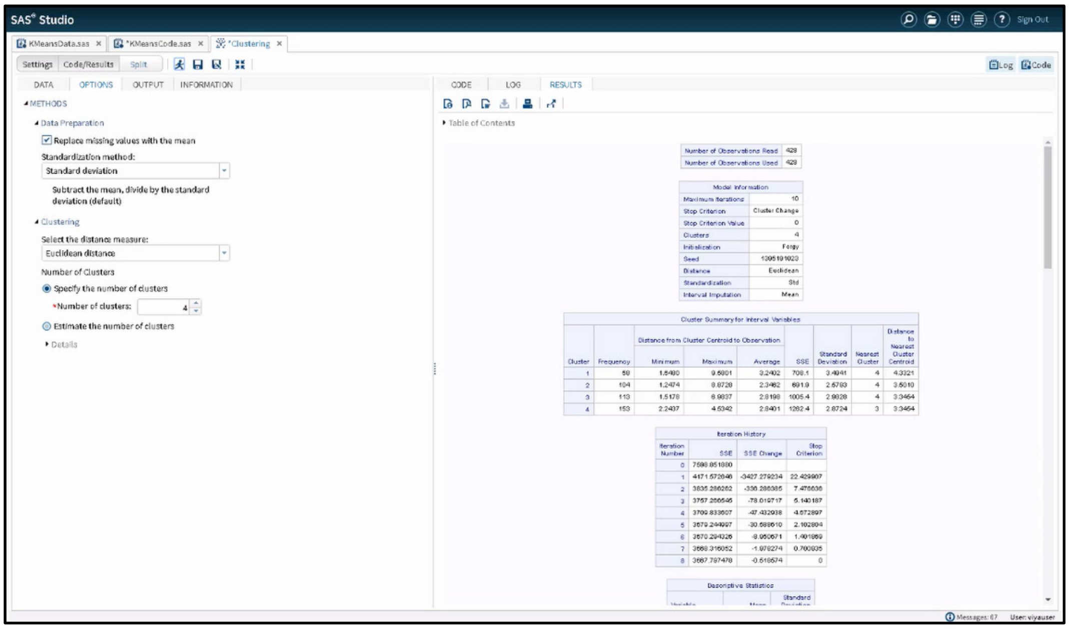
○ Estimate the number of clusters

Details

```
/*
 *
 * Task code generated by SAS Studio 4.1
 *
 * Generated on '10/13/16, 2:09 PM'
 * Generated by 'viyauser'
 * Generated on server [redacted]
 * Generated on SAS platform 'Linux LIN X64 2.6.32-642.4.2.el6.x86_64'
 * Generated on SAS version 'V.03.01M0P09072016'
 * Generated on browser 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Ge
 * Generated on web client [redacted]
 *
 */

ods noproctitle;

proc kclus data=MYCASLIB.CARS impute=mean standardize=std distance=euclidean
        maxclusters=4;
    input MSRP Invoice EngineSize Cylinders Horsepower MPG_City MPG_Highway Weight
        Wheelbase Length origin_asia origin_europe origin_usa drivetrain_all
        drivetrain_front drivetrain_rear / level=interval;
run;
```

Running the task creates four clusters with various descriptive model and cluster statistics on the Results tab, as shown in Figure 3.12.

Figure 3.12: Clustering Results



METHODS

Data Preparation

[✓] Replace missing values with the mean

Standardization method: Standard deviation

Subtract the mean, divide by the standard deviation (default)

Clustering

Select the distance measure: Euclidean distance

Number of Clusters

[✓ Specify the number of clusters]

*Number of clusters: 4

Estimate the number of clusters

Details

Number of Observations Read	428
Number of Observations Used	428

Model Information	
Maximum Iterations	10
Stop Criterion	Cluster Change
Stop Criterion Value	0
Clusters	4
Initialization	Forgy
Seed	1395191023
Distance	Euclidean
Standardization	Std
Interval Imputation	Mean

Cluster Summary for Interval Variables

Cluster	Frequency	Distance from Cluster Centroid to Observation			SSE	Standard Deviation	Nearest Cluster	Distance to Nearest Cluster Centroid
		Minimum	Maximum	Average				
1	58	1.6480	9.6801	3.2402	708.1	3.4941	4	4.3321
2	104	1.2474	8.8728	2.3462	691.9	2.6793	4	3.5010
3	113	1.5178	6.9837	2.8198	1005.4	2.9828	4	3.3464
4	153	2.2437	4.5342	2.8401	1282.4	2.8724	3	3.3464

Iteration History

Iteration Number	SSE	SSE Change	Stop Criterion
0	7598.851880		
1	4171.572646	-3427.279234	22.429907
2	3835.286262	-336.286385	7.476636
3	3757.266545	-78.019717	5.140187
4	3709.833607	-47.432938	4.672897
5	3679.244997	-30.588610	2.102804
6	3670.294326	-8.950671	1.401869
7	3668.316052	-1.978274	0.700935
8	3667.797478	-0.518574	0

Descriptive Statistics

Clustering Code in SAS Viya versus SAS 9

You can also perform clustering by coding the KCLUS procedure with various options. In this section, you will learn how to perform unsupervised clustering using PROC KCLUS, which is one of the SAS Visual Data Mining and Machine Learning statistical procedures in SAS Viya. If you have used PROC FASTCLUS in SAS 9 to perform k-means clustering, then keep reading to see how the new procedure compares.

For this example, we will use the data set INPDATA created from the SAS Viya documentation for PROC KCLUS. The names of the variables are generic, with the quantitative variables X and Y. To work with this data in SAS Viya, add a caslib and load the data into CAS using a DATA step, or whichever method you prefer.

SAS 9 Code

Before we look at the code for PROC KCLUS, let's consider Program 3.6, which is an example of k-means clustering using PROC FASTCLUS in SAS 9.

Program 3.6: SAS 9 Code

```
proc stdize data=work.inpData  ❶
          out=std_inpData
          method=range
          outstat=std_params;
   var x y;
run;

proc fastclus data=work.std_inpData
          maxcluster=3
          maxiter=10      ❷
          replace=random   ❸
          random=1
          out=work.scored;  ❹
   var x y;        ❺
   freq freq;      ❻
run;

proc stdize data=work.std_inpdata  ❼
          method=in (std_params)
          unstdize;
   var x y;
run;
```

❶ We often want to standardize input variables before clustering. In SAS 9, we have to do this outside of PROC FASTCLUS. We can use the STDIZE procedure to perform range standardization and then use PROC FASTCLUS on the standardized data.

❷ By default, PROC FASTCLUS uses Euclidean distance and a maximum of one iteration. We are asking for a maximum of 10 iterations in this code.

❸ Another default option is to use the first k complete data points as initial seeds for clustering. Here, however, I ask for random data values to act as cluster seeds, using the REPLACE=RANDOM option. In this code, I use a random seed of 1, using the option RANDOM=1.

❹ The OUT= option in PROC FASTCLUS requests an output data set containing all variables from the input data set, plus a cluster assignment and a distance to cluster variable.

❺ The VAR statement lists X and Y as the input variables.

❻ The FREQ statement names the variable containing the frequency represented by a row of data.

❼ After clustering is performed on the standardized data in PROC FASTCLUS, the clustering variables should be unstandardized. This can be done using PROC STDIZE one more time with options for unstandardizing.

SAS Viya Code

Let's look at Program 3.7 to see how PROC KCLUS can be programmed to perform a similar segmentation.

Program 3.7: SAS Viya Code

```
proc kclus data=mycaslib.inpData
          maxcluster=3
          standardize=range  ❶
          seed=1; ❷
   input x y; ❸
   freq freq; ❹
   score out=mycaslib.scored copyvars=(_all_); ❺
run;
```

❶ You don't need to standardize the data before you use the KCLUS procedure. Instead, you can request standardization using the STANDARDIZE option. The only options available are no standardization, range standardization, and standard deviation standardization.

❷ By default, PROC KCLUS selects initial seeds using random selection from the input data. Here we are using the same randomization seed of 1 that we used in the PROC FASTCLUS code.

❸ PROC KCLUS has an INPUT statement instead of a VAR statement.

❹ The same coding for the FREQ statement is used here as in the PROC FASTCLUS code.

❺ An output data set is created using a SCORE statement. To retain all of the variables from the input data set in the scored data set, use the COPYVARS= option.

The output of the PROC KCLUS code in SAS Viya looks very similar to the PROC FASTCLUS output. The frequencies of the observations assigned to each cluster are very similar to what we would get from PROC FASTCLUS using similar options.

You can produce plots using output data from PROC KCLUS and the same statistical graphics procedures that you might be familiar with in SAS 9, such as PROC SGPLOT, as we will see in the example in the next section. Be careful if you are working with very large data sets, because producing plots requires a great deal of processing.

PROC KCLUS Example

Let's look at the code in Program 3.8. This program uses PROC KCLUS to perform k-means clustering with Euclidean distance where missing values are imputed to the mean and inputs are standardized.

Program 3.8: PROC KCLUS

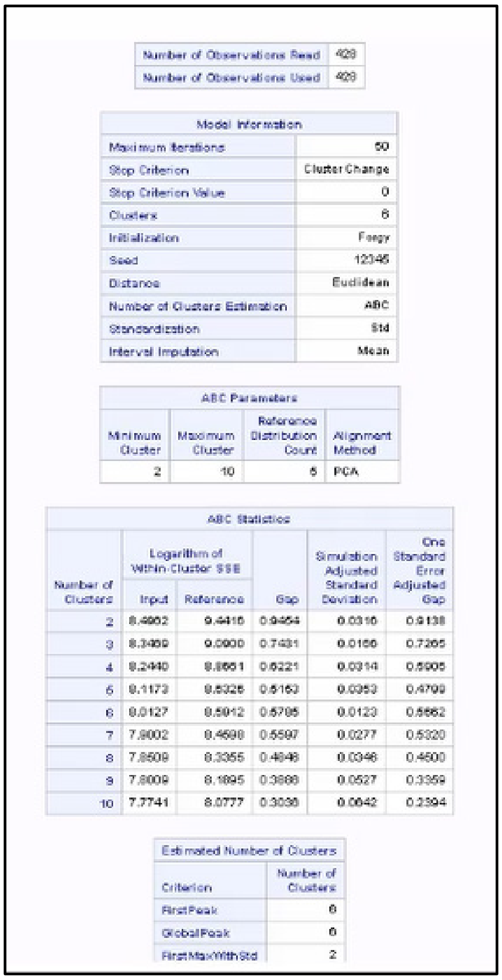
```
%let input vars=origin_: drivetrain_: msrp invoice enginesize
   cylinders horsepower mpg_city mpg_highway weigh
   Wheelbase length;
```

```
proc kclus data=mycaslib.cars seed=12345 standardize=STD
          impute=MEAN distance=EUCLIDEAN maxiters=50
          noc=ABC(b=5 criterion=all align=pca) maxclusters=10; ❶
   input &inputvars;
   ods output abcstats=mylib.clus_abcstats
          abcresults=mylib.clus_abcresults;
run;
```

❶ One of the main unknowns in the clustering algorithm is K, or the number of clusters to create. If you have previous experience with your data and know how many clusters to create, then specify that number using the MAXCLUSTERS= option. Otherwise, use the NOC=ABC or the aligned box criterion (ABC) for the initial estimate of K. The ABC techniques as additional parameters to fine-tune, so read the documentation for more details.

When using ABC to figure out K, the ABCResults and ABCStats ODS tables show the statistics used to choose the best K. The ABC statistics plot in Output 3.8 suggests the number of clusters that will be a good starting point.

Output 3.8: Results of Program 3.8



Number of Observations Read	428
Number of Observations Used	428

Model Information	
Maximum Iterations	50
Stop Criterion	Cluster Change
Stop Criterion Value	0
Clusters	6
Initialization	Forgy
Seed	12345
Distance	Euclidean
Number of Clusters Estimation	ABC
Standardization	Std
Interval Imputation	Mean

ABC Parameters

Minimum Cluster	Maximum Cluster	Reference Distribution Count	Alignment Method
2	10	5	PCA

ABC Statistics

Number of Clusters	Logarithm of Within-Cluster SSE		Gap	Simulation Adjusted Standard Deviation	One Standard Error Adjusted Gap
	Input	Reference			
2	8.4952	9.4416	0.9464	0.0316	0.9138
3	8.3469	9.0900	0.7431	0.0166	0.7265
4	8.2440	8.8661	0.6221	0.0314	0.5905
5	8.1173	8.6326	0.5153	0.0353	0.4799
6	8.0127	8.5912	0.5785	0.0123	0.5662
7	7.9002	8.4598	0.5597	0.0277	0.5320
8	7.8509	8.3355	0.4846	0.0346	0.4500
9	7.8009	8.1895	0.3886	0.0527	0.3359
10	7.7741	8.0777	0.3036	0.0642	0.2394

Estimated Number of Clusters

Criterion	Number of Clusters
FirstPeak	6
GlobalPeak	6
FirstMaxWithStd	2

You can also try different Ks for a more complete coverage or until you solve the business problem at hand. In Program 3.9, we will pick a specific K and rerun PROC KCLUS to get various cluster statistics that can be later used for generating plots.

Program 3.9

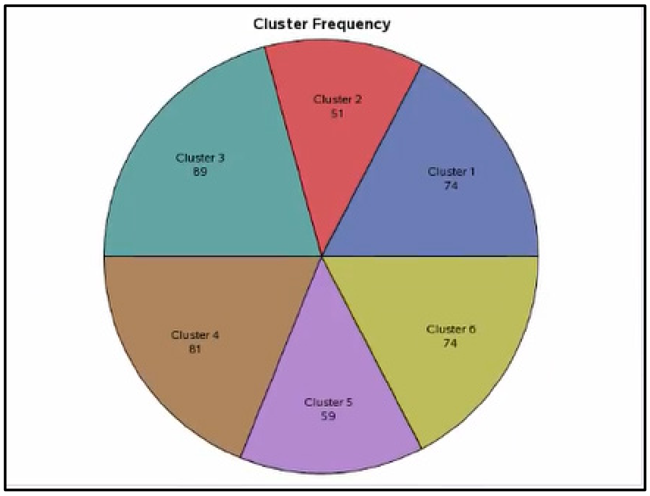
```
proc kclus data=mycaslib.cars
          standardize=STD impute=MEAN distance=EUCLIDEAN
          maxiters=50 maxclusters=6;
   input &input_vars;
   output out=mycaslib.cars_scored copyvars=(_all_);
```

```
    ods output clustersum=mylib.clus_clustersum;
    code file="&outdir/cluster1.sas"; ❶
run;

data mylib.clus_clustersum;
    set mylib.clus_clustersum;
    clusterLabel = catx(' ', 'Cluster', cluster);
run;

ods path work.templat(update) sasuser.templat(read)
            sashelp.tmplmst(read);

proc template;
    define statgraph simplepie;
    begingraph;
            entrytitle "Cluster Frequency";
            layout region;
            piechart category=clusterLabel response=frequency;
            endlayout;
    endgraph;
end;
run;

proc sgrender data=mylib.clus_clustersum tamplate=simplepie;
run;

proc sgpanel data=mycaslib.cards_scored;
    title "CarType in Clusters";
    panelby _cluster_id_ / layout=columnlattice columns=2;
    vbar type;
run;
title;
```

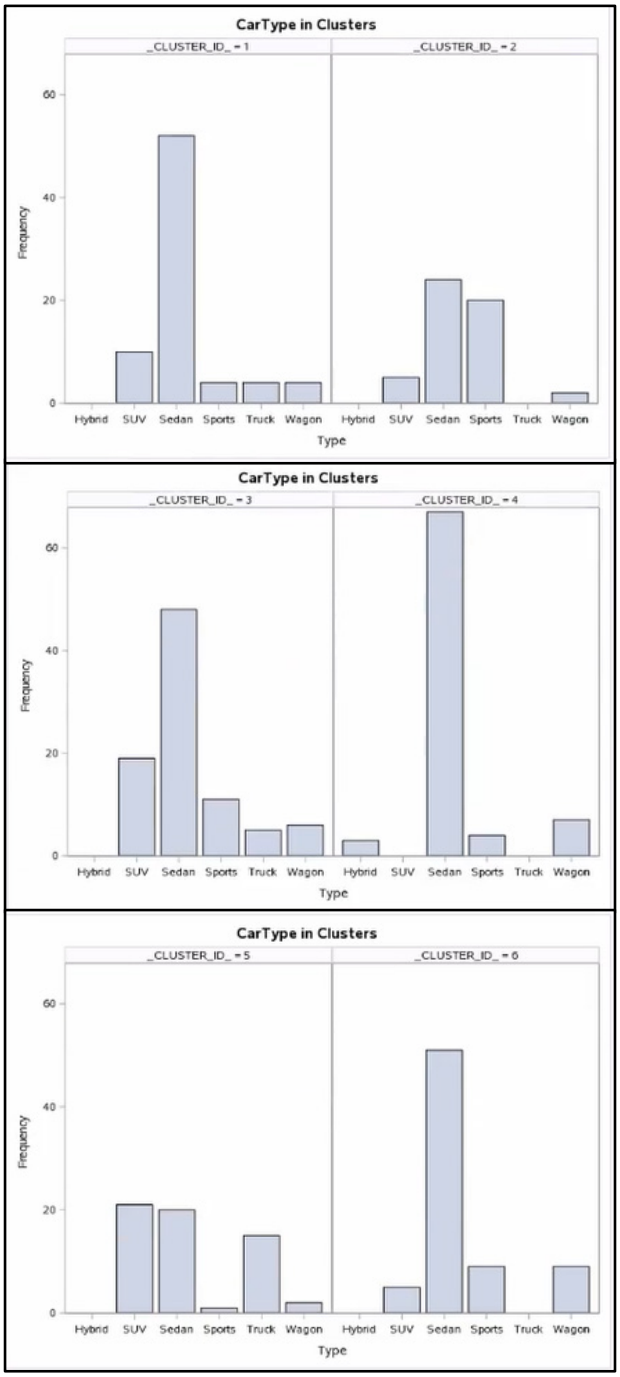
❶ The CODE statement saves SAS DATA step score code to the cluster1.sas file. This can later be used to score new data.

The pie chart in Output 3.9a shows the cluster frequency and the bar chart in Output 3.9b shows the profile of car types: hybrid, sedan, SUV, sports, truck, and so on, in each of the clusters.

Output 3.9a: Partial Results of Program 3.9



Output 3.9b: Partial Results of Program 3.9



You can draw similar plots to profile other characteristics, and you can re-adjust K until you are satisfied with the solution.

Principal Component Analysis

In this section, you will learn about principal component analysis (PCA), which falls under the umbrella of unsupervised learning. PCA is used to reduce the number of input variables or independent variables in your data. In other words, it performs variable reduction, sometimes called dimension reduction.

Let's look at this technique more closely with an example. For this example, we will use the cars data set in the SASHELP library. It contains information about cars, including their makes, models, manufacturers, and other characteristics. The first step is to load the cars data set from the SASHELP library into the MYCASLIB library. Note that dummy variables are created for the nominal inputs, Origin and DriveTrain.

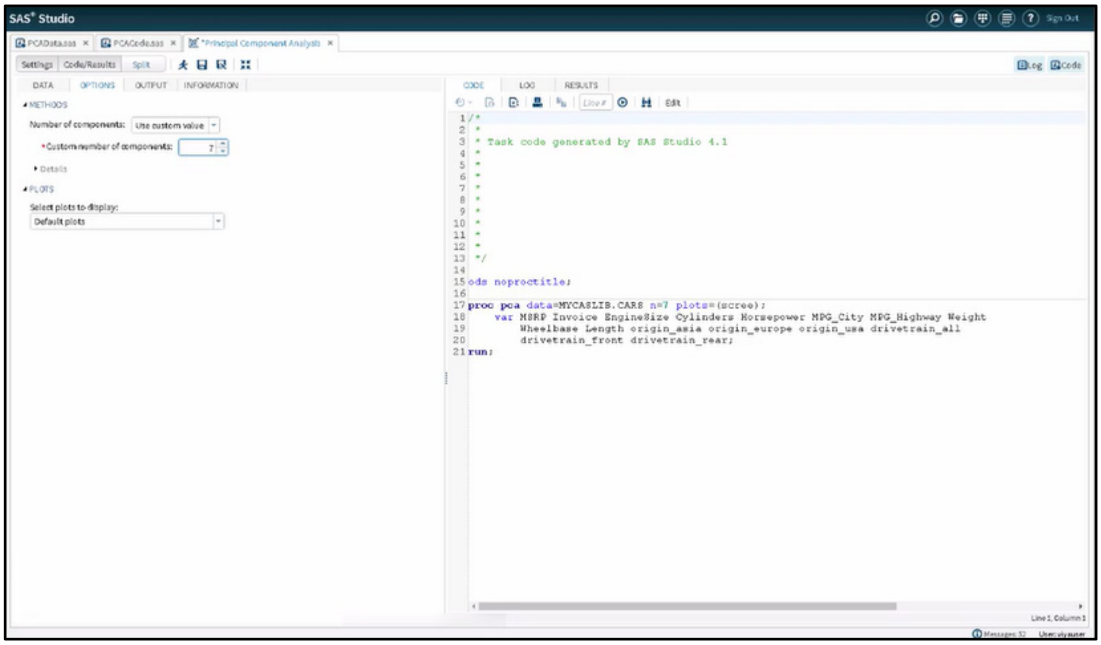
To perform PCA, you can use a SAS Studio task or run code. We will look at both methods, starting with the task.

Principal Component Analysis Task

Click the Task and Utilities tab on the left in SAS Studio. Expand Tasks and Unsupervised Learning. Double click on Principal Component Analysis to open the PCA window.

Select the cars data set in the MYCASLIB library, then select all available interval inputs in the data by clicking on the + sign next to the interval inputs window and selecting all variables. Go to the Options tab and perform these two actions as shown in Figure 3.13: select Use Custom Value from the Number of Components menu and then specify 7 for the Custom Number of Components. In SAS Studio, the code window on the right generates the necessary SAS code to perform PCA.

Figure 3.13: PCA Options Tab



METHODS

Number of components: Use custom value

*Custom number of components: 7

Details

PLOTS

Select plots to display: Default plots

```
/*
 *
 * Task code generated by SAS Studio 4.1
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */

ods noproctitle;

proc pca data=MYCASLIB.CARS n=7 plots=(scree);
    var MSRP Invoice EngineSize Cylinders Horsepower MPG_City MPG_Highway Weight
        Wheelbase Length origin_asia origin_europe origin_usa drivetrain_all
        drivetrain_front drivetrain_rear;
run;
```

Line 1, Column 1

Messages: 52 User: viyauser

Run the task by clicking the running person icon. This creates seven principal components with various descriptive and model statistics on the Results tab. The number 7 was chosen for this example because the first 7 principal components explain more than 95% of the variance in this data.

Principal Component Analysis Code in SAS Viya versus SAS 9

You can also perform principal component analysis by coding the PCA procedure with various options. In this section, you will learn how to perform unsupervised variable reduction using PROC PCA, which is one of the SAS Visual Data Mining and Machine Learning statistical procedures in SAS Viya. If you have used PROC PRINCOMP in SAS 9 to perform principal component analysis, then keep reading to see how the new procedure compares.

SAS 9 Code

In this example, we will use the Heart data set from the Sashelp library of data sets that are included in SAS. The numeric variables for the analysis are both demographic and health measures. Take a look at Program 3.10, which uses PROC PRINCOMP in SAS 9.

Program 3.10: SAS 9 Code

```
proc princomp data=sashelp.heart out=work.heart_pc plots=(pattern scree); ❶
   var AgeAtStart -- Smoking; ❷
run;
```

❶ The programming syntax is fairly simple. We name the data set and specify an output data set to contain our component scores.

❷ Here we list the input variables. The VAR statement uses a shorthand notation for the list of all variables in creation order from AgeAtStart to Smoking.

SAS Viya Code

Let's see how PROC PCA can be coded to perform the same analysis as the SAS 9 code. The PROC PCA code in Program 3.11 is very similar to the PROC PRINCOMP code.

Program 3.11: SAS Viya Code

```
proc pca data=mycaslib.heart plots=all; ❶
   var AgeAtStart -- Smoking; ❷
   output out=mycaslib.heart_pc copyvars=(_all_);❸
   code file="c:\mydir\scorecode.sas"; ❹
run;
```

❶ We read the Heart data from our active caslib. We can ask for a scree plot and a pattern profile plot using the PLOT=ALL option.

❷ The VAR statement is identical to the one used in the SAS 9 code.

❸ One difference between PROC PCA and PROC PRINCOMP is that the output data set is requested in an OUTPUT statement in PROC PCA, whereas it is requested in the PROC statement in PROC PRINCOMP. Another difference is that by default, PROC PCA does not include variables from the input data set. However, they can be requested using the COPYVARS option.

❹ In addition, we are including a CODE statement. The CODE statement produces a file of code for creating component scores for new data in a SAS DATA step. This file must be created manually using output from PROC PRINCOMP.

The output from PROC PCA looks very much like PROC PRINCOMP output. You will get a table of simple statistics and a correlation matrix table. Next, you will see the scree plot, followed by the eigenvalues table and the table of eigenvectors. The pattern profile plot is the same as the one produced by PROC PRINCOMP.

It is worth noting that although PROC PRINCOMP is limited to principal component analysis using the method of eigen decomposition, PROC PCA allows for three other methods as well. Another difference is that PROC PCA can produce only a limited set of plots compared with PROC PRINCOMP. However, you can produce other plots using output data sets and statistical graphics procedures, such as PROC SGPLOT.

Principal Component Analysis Example

Let's look another example of how to perform PCA and interpret the results in SAS Viya with the code in Program 3.12. This example uses the cars data set from the SASHELP library.

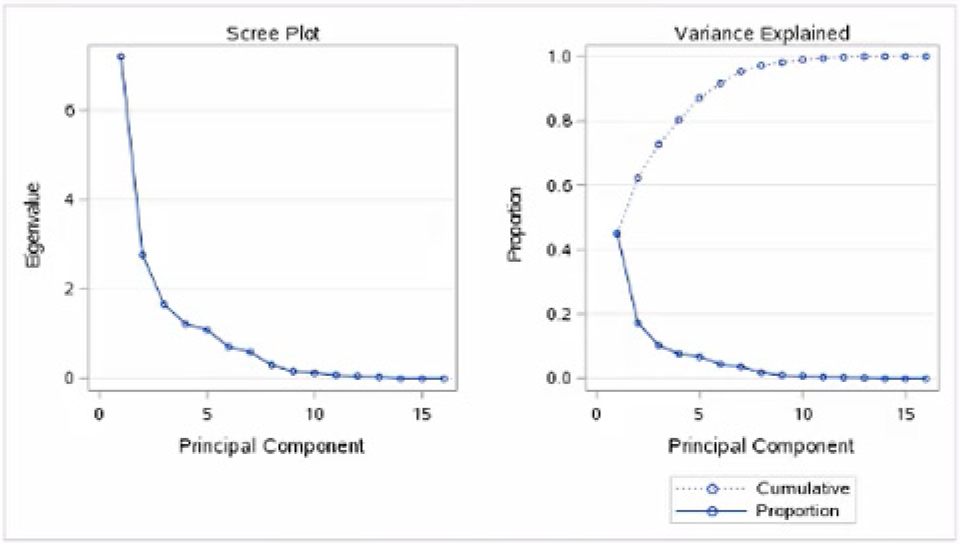
Program 3.12: PCA code

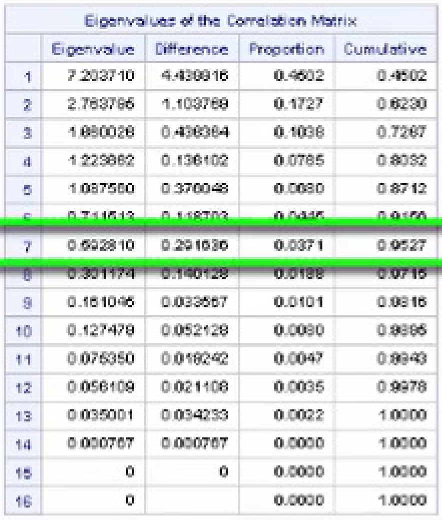
```
proc pca data=mycaslib.cars /*COV*/
          prefix=PC method=EIG plots=all; ❶

   var &input_vars;

run;
```

❶ This statement uses PROC PCA to perform principal component analysis using Eigen decomposition of the correlation matrix. The PLOTS= option draws various plots including the scree and variance explained plots.

Using the scree plot and the eigenvalues output table, you can decide how many principal components to keep. In this example, we will choose seven principal components because they explain 95% of the cumulative variance in the data, as shown in Output 3.12.

Output 3.12: Partial Results of Program 3.12





Eigenvalues of the Correlation Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
1	7.203710	4.439916	0.4502	0.4502
2	2.763796	1.103769	0.1727	0.6230
3	1.660026	0.436164	0.1038	0.7267
4	1.223862	0.136102	0.0765	0.8032
5	1.087760	0.376048	0.0680	0.8712
6	0.711513	0.118703	0.0445	0.9156
7	0.592810	0.291636	0.0371	0.9527
8	0.301174	0.140128	0.0188	0.9715
9	0.161046	0.033567	0.0101	0.9816
10	0.127479	0.052128	0.0080	0.9896
11	0.075350	0.018242	0.0047	0.9943
12	0.056109	0.021108	0.0035	0.9978
13	0.035001	0.034233	0.0022	1.0000
14	0.000767	0.000767	0.0000	1.0000
15	0	0	0.0000	1.0000
16	0		0.0000	1.0000

What Program 3.12 accomplished was to reduce the original 16 input variables to 7 without much loss of information.

In Program 3.13, we will re-run PCA, this time to create only 7 principal components.

Program 3.13: Modified PCA code

```
proc pca data=mycaslib.cars /*COV*/
          prefix=PC method=EIG plots=all n=7; ❶

   var &input_vars;

   output out=mycaslib.cars_scored copyvars=(_all_) score=PC_;
   code file="&outdir/pca1.sas"; ❷
run;
```

❶ Adding the N= option creates only n number of principal components.

❷ The CODE statement saves SAS DATA step score code to pca1.sas, a file that can be used to score new data later.

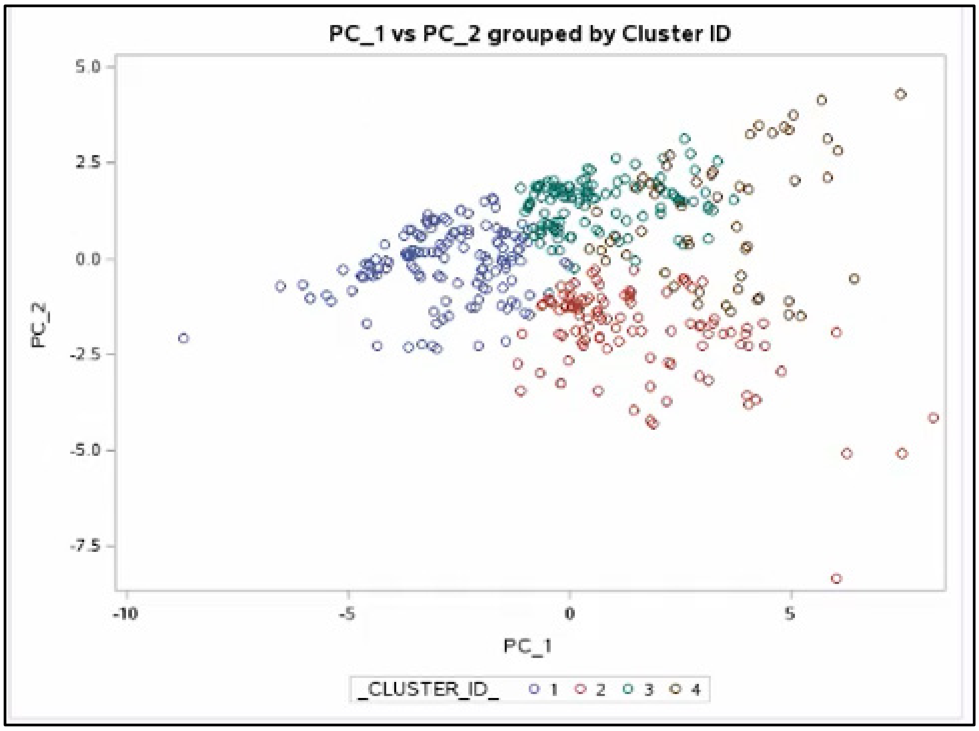
Clustering and PCA Visualization Example

Program 3.14 creates a visualization (Output 3.14) that combines the PCA results and clustering to show a scatterplot of principal component 1 versus principal component 2 while color coding with the cluster ID. This brings the data down to two primary dimensions and colors each data point depending on the cluster it belongs to. This is a neat exploration plot that ties the two unsupervised techniques together.

Program 3.14: Clustering and PCA Visualization

```
proc kclus data=mycaslib.cars_scored standardize=STD impute=MEN
           distance=EUCLIDEAN maxiters=50 maxclusters=4;

    input &input_vars;

    output out=mycaslib.cars_scored copyvars=(_all_);

run;

proc sgplot data=mycaslib.cars_scored(keep=PC_1 PC_2 _cluster_id_);
    title "PC_1 vs PC_2 grouped by Cluster ID";
    scatter x=PC1 y=PC_2 / group=_cluster_id_;
run;
title;
```

Output 3.14: Partial Results of Program 3.14



Supervised Learning

In this section, you will learn how to use SAS Viya to perform supervised learning tasks. We will not be looking at the tasks in SAS Studio; instead, we will focus on the differences between the new SAS Viya procedures and the previous SAS 9 code. Here are the procedures that will be discussed in this section:

Name	SAS Studio Task	SAS Viya Procedure	SAS 9 Procedure
Linear Regression Modeling:	Linear regression	PROC REGSELECT	PROC GLMSELECT PROC REG
Logistic Regression Modeling	Logistic Regression	PROC LOGSELECT	PROC LOGISTIC
Generalized Linear Models	Generalized Linear Models	PROC GENSELECT	PROC GENMOD

Linear Regression

In this section, you will learn how to build regression models using PROC REGSELECT, which is one of the SAS Visual Data Mining and Machine Learning statistical procedures in SAS Viya. If you have used PROC REG or PROC GLMSELECT in SAS 9 to perform linear regression, then keep reading to see how the new procedure compares.

For this example, we will use the simulated data set ANALYSISDATA created from the SAS Viya documentation. The names of the variables are generic, with the quantitative variables X1 through X20 and categorical inputs C1 through C3. The target, or response variable, is named Y. We want to fit and validate a model for a target, Y, using training and validation data. PROC REGSELECT will partition the data into training and validation samples. PROC GLMSELECT in SAS 9 has the same functionality. However, PROC REG does not have a CLASS statement for creating dummy variables for categorical inputs, nor does it have functionality for using validation data. Let's compare some code for a predictive regression model in PROC GLMSELECT in SAS 9 with equivalent code in PROC REGSELECT in SAS Viya.

Program 3.15: SAS 9 Code

```
proc glmselect data=work.analysisData;❶
   partition role=Role(validate='VAL'
                    test='TEST'); ❷
   class c1 c2 c3 / param=glm; ❸
   model y =  c1|c2|c3|x1|x2|
              x3|x4|x5|x6|x7|
              x8|x9|x10|x11|x12|
              x13|x14|x15|x16|x17|
              x18|x19|x20 @2 / ❹
   selection = stepwise
              (select=sl
               stop=sl
               slentry=0.1
               slstay=0.1
               choose=validate)
        hierarchy=single; ❺
 run;
```

Program 3.16: SAS Viya Code

```
proc regselect data=mycas.analysisData;❶
   partition role=Role(validate='VAL'
                    test='TEST'); ❷
   class c1 c2 c3 / param=GLM; ❸
   model y =  c1|c2|c3|x1|x2|
              x3|x4|x5|x6|x7|
              x8|x9|x10|x11|x12|
              x13|x14|x15|x16|x17|
              x18|x19|x20 @2; ❹
   selection method = stepwise
              (select=sl
               stop=sl
               slentry=0.10
               slstay=0.10
               choose=validate)
        hierarchy=single; ❺
 run;
```

❶ The PROC statements use only a DATA= option and are identical.

❷ The PARTITION statements use exactly the same options for partitioning the data, in this case using a variable named Role in the data set.

❸ The CLASS statements are identical. The options for parameterizations are also the same.

❹ The MODEL statements start out identical as well. The model we are specifying includes all main effects and all two-way interactions. One difference is how the selection criteria are specified in the two procedures. In PROC GLMSELECT, the selection options are requested in the MODEL statement, whereas in PROC REGSELECT, they are selected in a separate statement – the SELECTION statement.

❺ In PROC REGSELECT, the SELECTION statement for selection, stopping, significance levels, and choosing the final model, are the same as the MODEL statement selection options in PROC GLMSELECT.

After you run Program 3.16 to get the results of the PROC REGSELECT procedure, at the top of the results, you will see information about the model, the selection options, the number of observations in each data partition, the CLASS statement variables, and the number of effects and parameters. A selection summary comes next. By default, no details are displayed for each step in the selection, although those details can be requested. Eighteen effects were entered based on the 0.1 significance level criterion. The asterisk by the Validation Average Squared Error value shows that the model at step 10 has the lowest value and is, therefore, the winner.

The next set of tables shows details about the selected model. The information is similar to the information provided by PROC GLMSELECT. The last table in the PROC REGSELECT output is a task timing table. This table is not in the PROC GLMSELECT output. Because SAS Viya is intended for use with very large data sets, this information might be of interest to you.

In SAS Viya, plots and diagnostic plots are not available within the procedures themselves. However, you can produce plots using output data from PROC REGSELECT and the same statistical graphics procedures that you might have used in SAS 9, such as PROC SGPLOT.

Logistic Regression

In this section, you will learn how to perform logistic regression model building using PROC LOGSELECT, which is one of the SAS Visual Data Mining and Machine Learning statistical procedures in SAS Viya. If you have used PROC LOGISTIC in SAS 9 to perform logistic regression, then keep reading to see how the new procedure compares.

For this example, we will use the simulated data set GETSTARTED created from the SAS Viya documentation. The names of the variables are rather generic, with quantitative variables X1 through X10 and one categorical input, C. The target, or response variable, is named Y and it is binary, coded as 0 or 1.

In contrast to PROC LOGISTIC in SAS 9, PROC LOGSELECT, like all procedures in SAS Viya, can perform calculations in multiple threads and is designed to run on a cluster of machines that distributes the data and calculation. However, the programming and results are very similar between the two.

Let's look at an example of coding for a logistic regression model using PROC LOGISTIC and code that produces similar results in PROC LOGSELECT. We want to build a model using forward selection that predicts Y using both quantitative and categorical inputs.

Program 3.17: SAS 9 Code

```
proc logistic data=work.getstarted; ❶
   class C(param=GLM); ❷
   model y(event='1') =  C X1-X10 /
              selection=forward; ❸
run;
```

Program 3.18: SAS Viya Code

```
proc logselect data=mycaslib.getstarted; ❶
   class C; ❷
   model y(event='1')= C x1-x10;
   selection method=forward(select=1
                  stop=s1 slentry=0.05);❸
run;
```

❶ We are not requesting any options in either of the PROC statements.

❷ The CLASS statements in the two procedures are similar. All parameterizations available in one PROC are available in the other. The difference is that PROC LOGISTIC uses effects parameterization by default, whereas PROC LOGSELECT uses GLM parameterization, a less than full rank parameterization by default.

❸ Model selection in the MODEL statements are the same. Both assume a logit link and binomial probability distribution when the target variable has only two levels. PROC LOGSELECT models only binary target variables. Model selection is coded somewhat differently in the two procedures. In PROC LOGISTIC, it is requested after a slash in the MODEL statement. In PROC LOGSELECT, model selection options are requested in a SELECTION statement. A major difference between the two procedures is that PROC LOGISTIC is limited to using p-value criteria for sequential selection, where PROC LOGSELECT also allows selection based on information criteria and LASSO. In addition, PROC LOGSELECT allows the user to partition the data set into training, validation, and test data sets, and to use validation error to select the best model in a sequence. To request the p-value criteria for model selection in the SELECTION statement, we add suboptions to the METHOD=FORWARD option. Those are SELECT=SL and STOP=SL, along with SLENTRY=.05, which matches PROC LOGISTIC's default entry criterion.

After you submit the code for PROC LOGSELECT, you can look at the results. The descriptive information in the first five tables is identical to the information displayed in PROC LOGISTIC. Model selection details are reported in the Selection Summary table. By default, no details are displayed for each step, although the details can be requested.

Two effects entered based on the 0.05 p-value criterion. PROC LOGSELECT reports only one test of the global null hypothesis, the Likelihood Ratio test. PROC LOGISTIC adds the Score and Wald tests. PROC LOGSELECT adds the AICC to the list of fit statistics provided by PROC LOGISTIC.

The Parameter Estimates table is identical to the one that PROC LOGISTIC produces. The last table in the PROC LOGSELECT output is a task timing table. This table is not in the PROC LOGISTIC output. Because SAS Viya is intended for use with very large data sets, this information might be of interest to you.

A few more differences between PROC LOGSELECT and PROC LOGISTIC are helpful to know about. Because SAS Viya procedures are used with very large data sets, fewer plots are available within the statistical procedures. However, you can produce plots using output data from PROC LOGSELECT and the same statistical graphics procedures you might have used in SAS 9, such as PROC SGPLOT. Also, PROC LOGSELECT does not include options for post-fitting, such as ESTIMATE and CONTRAST statements. PROC LOGSELECT, compared with PROC LOGISTIC, also uses different optimization methods by default.

Generalized Linear Models

In this section, you will learn how to estimate generalized linear models using PROC GENSELECT, which is one of the SAS Visual Data Mining and Machine Learning statistical procedures in SAS Viya. If you are familiar with using PROC GENMOD in SAS 9, then keep reading to see how the new procedure compares.

For this example, we will use the simulated data set GETSTARTED created from the SAS Viya documentation. The names of the variables are rather generic, with five categorical inputs C1 through C5. The target, or response variable, is named Y and it represents counts.

PROC GENSELECT, like all procedures in SAS Viya, can perform calculations in multiple threads and is designed to run on a cluster of machines that distributes the data and calculations. However, the programming and results are very similar to what you are used to in PROC GENMOD in SAS 9.

Before we start looking at the code for PROC GENSELECT, let's look at an example of coding for a Poisson regression model using PROC GENMOD, and then see how they compare. We want to build a model for predicting Y, a count variable, using all of the categorical inputs.

Program 3.19: SAS 9 Code	Program 3.20: SAS Viya Code

`proc genmod data=work.getStarted;` ❶
`   class C1-C5;` ❶
`   model Y = C1-C5 /`
`   Dist=Poisson Link=Log;` ❷
`run;`
	`proc genselect data=mycaslib.getstarted;` ❶
	`   class C1-C5;` ❶
	`   model Y = C1-C5 /`
	`   Distribution=Poisson Link=Log;` ❷
	`run;`

❶ The CLASS statements in the two procedures are identical. All parameterizations available in one procedure are available in the other. We are using the default parameterization – GLM parameterization – a less than full rank parameterization. This is also the default in PROC GENMOD.

❷ The MODEL statements are nearly identical. A slight difference is that the distribution option in PROC GENMOD is DIST, whereas you can use the full name, DISTRIBUTION, in PROC GENSELECT

After you submit the code for PROC GENSELECT, the first three tables in the results, down to the Class Level information table, are nearly identical to those displayed from PROC GENMOD, and they report descriptive information. The Dimensions table comes next and describes the design matrix. The Fit Statistics table is somewhat different from the one that's produced by PROC GENMOD. There are no reported deviance or Pearson chi-square fit statistics.

PROC GENMOD reports the value of the log likelihood and the full log likelihood, whereas PROC GENSELECT reports the -2 log likelihood. The -2 log likelihood is simply -2 times the value of the full likelihood reported in PROC GENMOD.

The fit statistics tables from both procedures report the information criteria, AIC, AICC, and SBC. The Parameter Estimates table from PROC GLMSELECT does not show the confidence limits for the parameter estimates, and it displays the parameter values differently than the table from PROC GENMOD. Otherwise, the tables show the same results except for the task timing table.

As the name implies, PROC GENSELECT can also be used for model selection. PROC GENMOD doesn't have that functionality. In PROC GENSELECT, you can use many different selection methods and stopping criteria for arriving at a best subset model. These include stepwise methods based on p-values or fit statistics and final selection based on average square error in the validation sample.

PROC GENSELECT can also be used to partition a data set into Training, Validation, and Test data sets. A few more differences between PROC GENSELECT and PROC GENMOD are helpful to know about. In SAS Viya, plots are not available within the procedures themselves. However, you can produce plots using output data from PROC GENSELECT and the same statistical graphics procedures that you might have used in SAS 9, such as PROC SGPLOT. Also, PROC GENSELECT does not include options for post-fitting such as ESTIMATE and CONTRAST statements.

Resources

This chapter is based on the "An Introduction to SAS Viya Programming for SAS 9 Programmers" videos in "SAS® Viya® Enablement," a free course available from SAS Education.

You may find the following documentation helpful as you learn more about programming in SAS® Viya®:

- SAS Viya Data Mining and Machine Learning: Procedures Guide
- SAS® Studio 3.71: User's Guide

Chapter 4: Data Management in SAS Viya

Introduction

In this chapter you will learn about several applications that you can use to manage your data in SAS Viya. First, we will look at SAS Data Explorer, which enables you to load, import, and profile your data. Then, we will discuss SAS Data Studio, which helps you cleanse, prepare, and transform your data. Finally, we will explore SAS Lineage Viewer, which enables you to manage and govern data assets and their relationships.

SAS Data Explorer

SAS Data Explorer enables you to copy data to memory on SAS Cloud Analytic Services (CAS) server and to perform related tasks. There are two versions of SAS Data Explorer: a stand-alone web application and a window that can be displayed from SAS Viya applications.

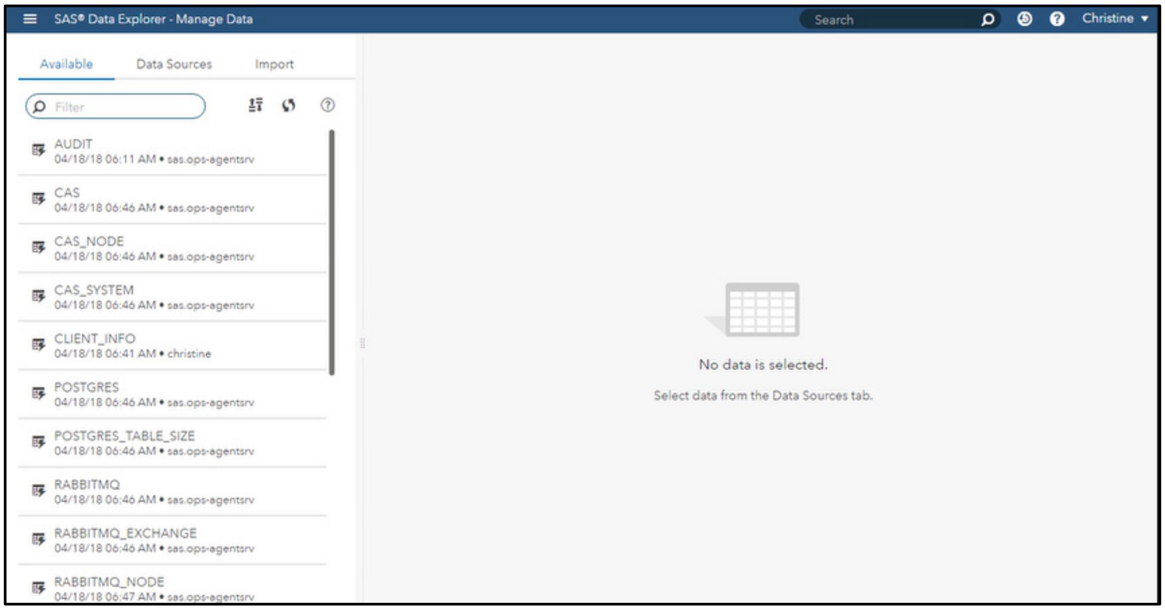
Connect to Data

In this section, you will learn how to use SAS Data Explorer to manage and discover data in the SAS Viya environment using related tasks such as loading and analyzing data.

When you log into SAS Data Explorer through the Manage Data action, you will see the Choose Data window. On the left, there are three tabs, as shown in Figure 4.1:

- **Available** – the Available tab shows you a list of all the tables that have been loaded into CAS memory.
- **Data Sources** – the Data Sources tab shows you all of the CAS servers and data connections available to you in your SAS Viya environment.
- **Import** – the Import tab enables you to import local files from your browser; social media such as Twitter, Facebook, Google Analytics, or YouTube feeds; and Geo Enrichment data from Esri.

Figure 4.1: Choose Data Window


SAS® Data Explorer - Manage Data

Available | Data Sources | Import

AUDIT
04/18/18 06:11 AM • sas.ops-agentsrv

CAS
04/18/18 06:46 AM • sas.ops-agentsrv

CAS_NODE
04/18/18 06:46 AM • sas.ops-agentsrv

CAS_SYSTEM
04/18/18 06:46 AM • sas.ops-agentsrv

CLIENT_INFO
04/18/18 06:41 AM • christine

POSTGRES
04/18/18 06:46 AM • sas.ops-agentsrv

POSTGRES_TABLE_SIZE
04/18/18 06:46 AM • sas.ops-agentsrv

RABBITMQ
04/18/18 06:46 AM • sas.ops-agentsrv

RABBITMQ_EXCHANGE
04/18/18 06:46 AM • sas.ops-agentsrv

RABBITMQ_NODE
04/18/18 06:47 AM • sas.ops-agentsrv

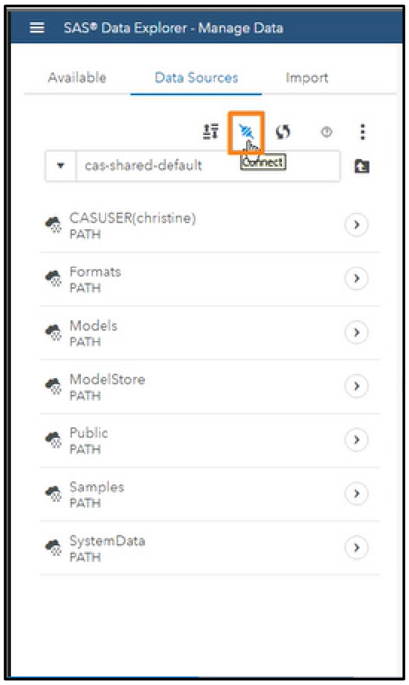
No data is selected.

Select data from the Data Sources tab.

Create a Data Connection to a File

In order to create a data connection, click on the Connect icon on the Data Sources Tab, as shown in Figure 4.2.

Figure 4.2: Connect Icon


SAS® Data Explorer - Manage Data

Available | Data Sources | Import

Connect

cas-shared-default

CASUSER(christine)
PATH

Formats
PATH

Models
PATH

ModelStore
PATH

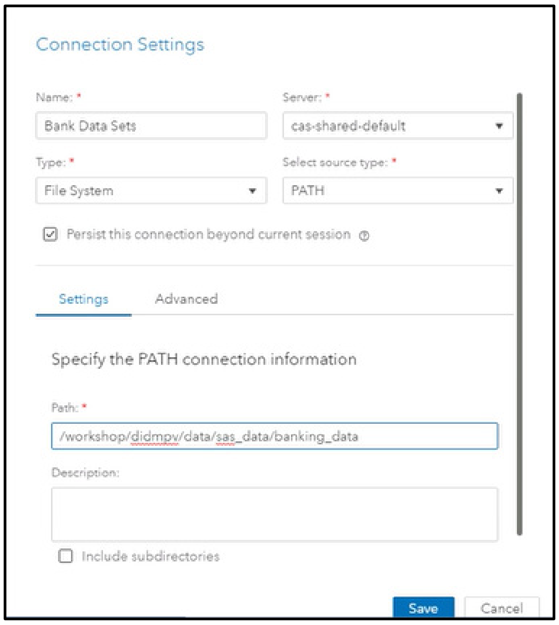
Public
PATH

Samples
PATH

SystemData
PATH

You will be presented with a Connection Settings wizard as shown in Figure 4.3. In the Type field, select “File System,” and then choose the source type. If you want your connection to persist after you leave the SAS Viya environment, select “Persist this connection beyond the current session.”

Figure 4.3: Connection Settings Wizard – File System


Connection Settings

Name: * Bank Data Sets

Server: * cas-shared-default

Type: * File System

Select source type: * PATH

☑ Persist this connection beyond current session

Specify the PATH connection information

Path: * /workshop/didmpv/data/sas_data/banking_data

Description:

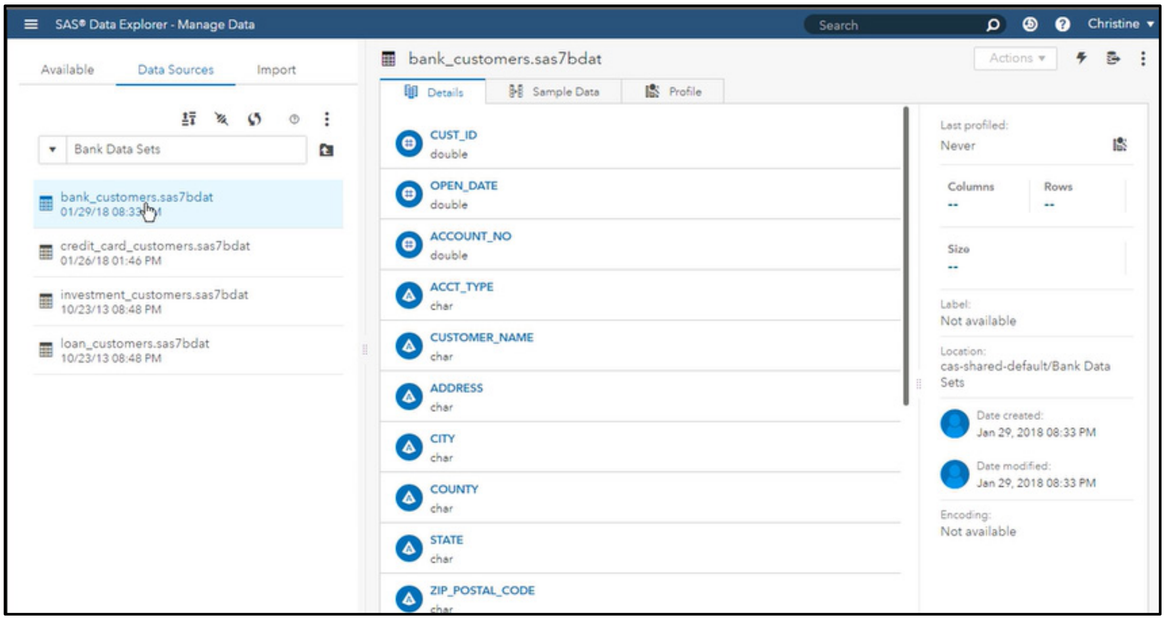
☐ Include subdirectories

Then, specify a Path that is local to your CAS Server where you can access the data. Click "Test Connection" to make sure that the CAS Server can access that location. If the connection is successful, you can hit Save. When you return to the Data Sources tab, the data set connection should be added under the server that you specified.

If you navigate into that connection by clicking on the arrow at the right, you can see the data sets in that file system path location. A location can include SAS data sets, text files, Excel spreadsheets, and delimited files.

When you click on a data set to select that table, you will be presented with the table information on a Details tab that gives you the metadata about the columns including the names of the columns and their data types, as shown in Figure 4.4.

Figure 4.4: Details



bank_customers.sas7bdat 01/29/18 08:33 PM
credit_card_customers.sas7bdat 01/26/18 01:46 PM
investment_customers.sas7bdat 10/23/13 08:48 PM
loan_customers.sas7bdat 10/23/13 08:48 PM

bank_customers.sas7bdat

CUST_ID	double
OPEN_DATE	double
ACCOUNT_NO	double
ACCT_TYPE	char
CUSTOMER_NAME	char
ADDRESS	char
CITY	char
COUNTY	char
STATE	char
ZIP_POSTAL_CODE	char

Last profiled: Never

Columns: --
Rows: --
Size: --

Label: Not available

Location: cas-shared-default/Bank Data Sets

Date created: Jan 29, 2018 08:33 PM

Date modified: Jan 29, 2018 08:33 PM

Encoding: Not available

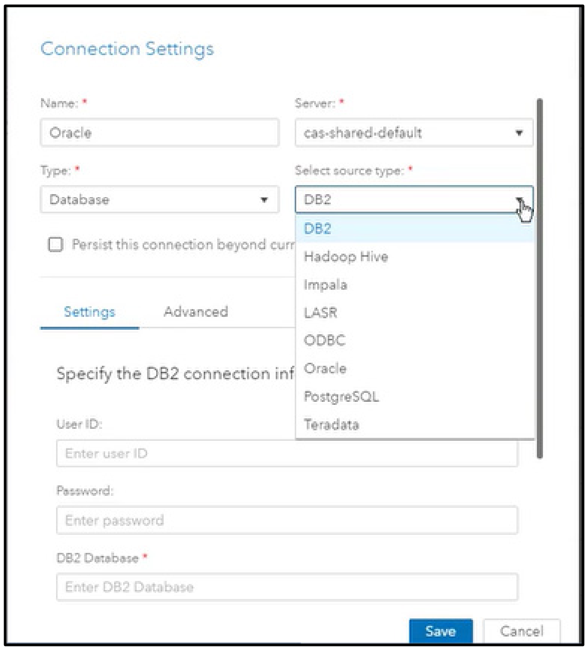
If you select the Sample Data tab, you will see a sample of 100 rows of data so that you can review and see what your data looks like and notice any inconsistencies that may need to be addressed before it can be used for analytics. The Profile tab enables you to generate some basic metrics about the table.

Create a Data Connection to Database Tables

In the previous section, you learned how to connect to local files into the SAS Viya environment. Now we will learn how to connect to relational database tables. Click on the Data Sources tab to bring up the Connection Settings wizard to create a connection, as shown earlier.

In the Type field on the wizard, select "Database." In the Select source type field, you will see a list of databases. Choose the one that corresponds to the type of database that you want to import, as shown in Figure 4.5. For example, to select an Oracle database, select Oracle from the list.

Figure 4.5: Connection Settings Wizard – Database



Connection Settings

Name: * Oracle

Server: * cas-shared-default

Type: * Database

Select source type: * DB2

Persist this connection beyond curr

Specify the DB2 connection inf

User ID: Enter user ID

Password: Enter password

DB2 Database * Enter DB2 Database

At the bottom of the wizard, fill out the fields to add your connection information. The connection information is unique to each database. If you select another database from the list, then you will get connection information specific to that database access.

After you have added your connection information, click "Test Connection." Once the connection is successful, click "Save." When you return to the Data Sources tab, you will see your new database connection listed.

When you double-click on the database connection, you will see the tables that are in your schema. Clicking on a table will give you the option to view basic details about the columns and their data types as well as sample data of the first 100 rows.

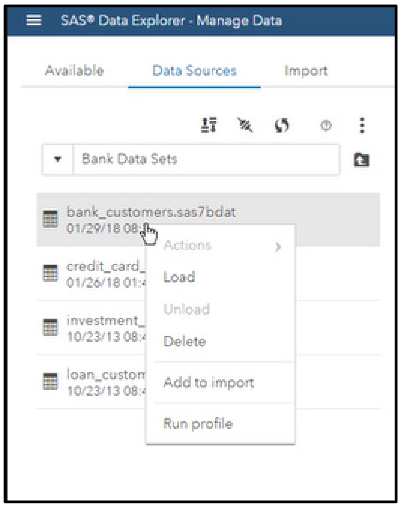
Import Data

Now that you have created a connection to your data, we will learn how to import a SAS data sets and local files into the SAS Viya environment and load them into memory.

Import SAS Data Sets

In the Data Sources tab, right-click on the SAS table that you want to import and select "Add to Import" as shown in Figure 4.6. If you just want to load the data into CAS memory and not actually load the table and data into the SAS Viya environment in a distributed format, you can select "Load" instead.

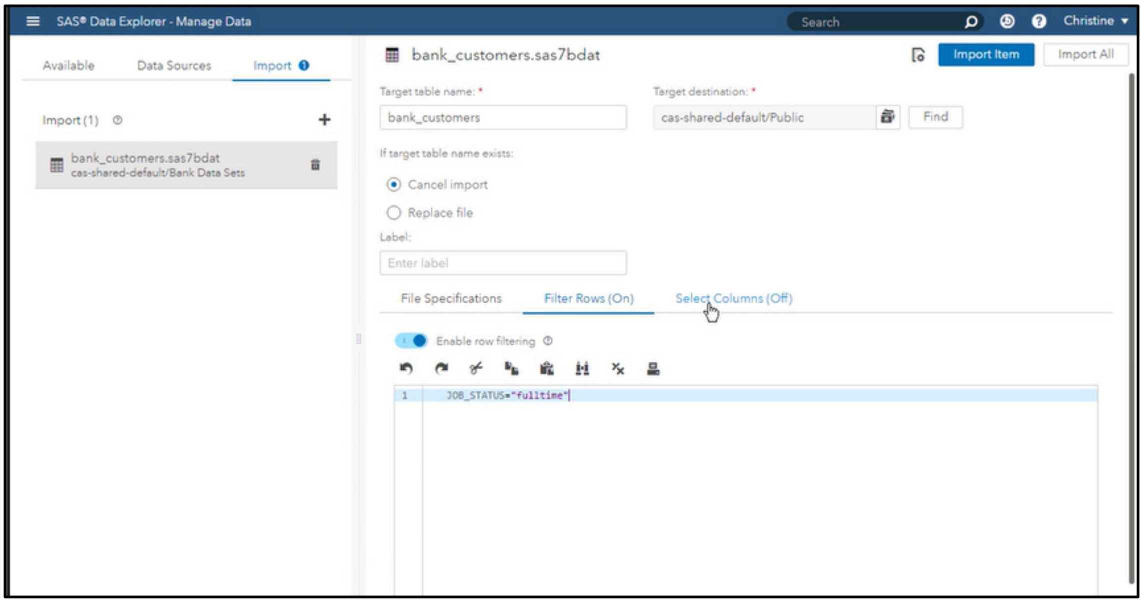
Figure 4.6: Import Table



After you select "Add to Import," you will see the Import window on the right side of your screen. You can specify your target location in the CAS environment where you want the distributed data to reside by using the Target Destination option. You can change the Target table name as well. If you want to run this as a job later on, you can select "Replace file" so that you can create a job and schedule it for later.

Because we are loading a SAS data set, you will get certain file specifications such as encryption information. You can also use the Filter Rows so that instead of bringing in all of the data, you apply filters to the data coming in as shown in Figure 4.7.

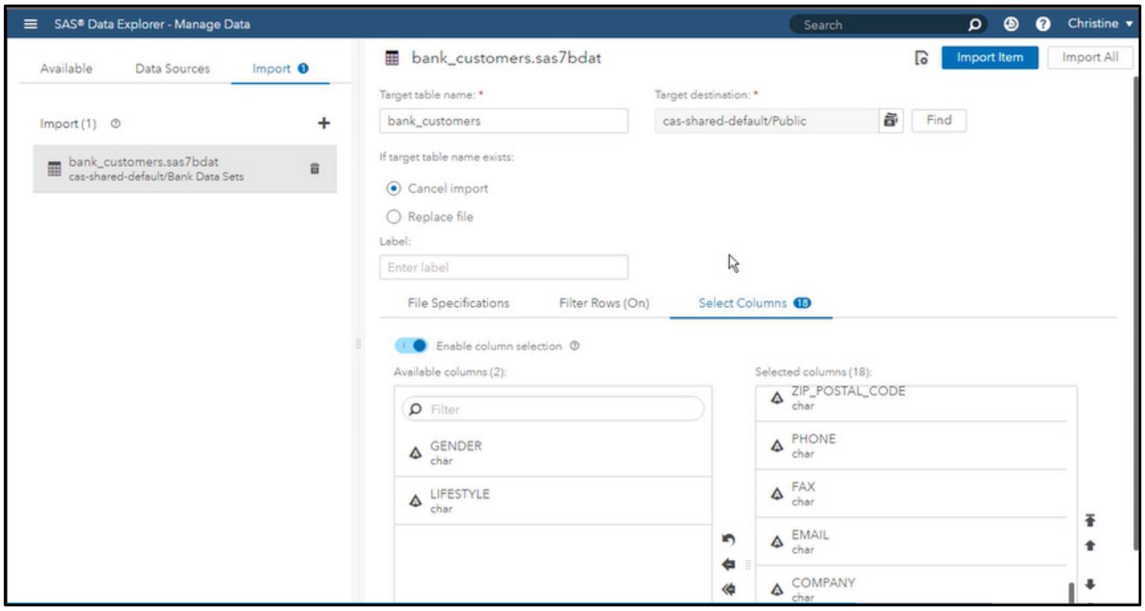
Figure 4.7: Import tab – Filter Rows



```
JOB_STATUS="fulltime"
```

You can also select the column filter if you enable column selection in the "Select Columns" tab. By default, all of the columns are selected. You can double-click columns or use the navigation arrows to pull columns back to the available columns area, which removes them from the import, as shown in Figure 4.8. The number of columns that you will be importing is indicated in the "Select Columns" header.

Figure 4.8: Import Tab – Select Columns



bank_customers.sas7bdat
cas-shared-default/Bank Data Sets

bank_customers.sas7bdat

Target table name: * bank_customers

Target destination: * cas-shared-default/Public

If target table name exists:
[✓ Cancel import]
Replace file

Label: Enter label

Enable column selection [✓]

Available columns (2):

GENDER char
LIFESTYLE char

Selected columns (18):

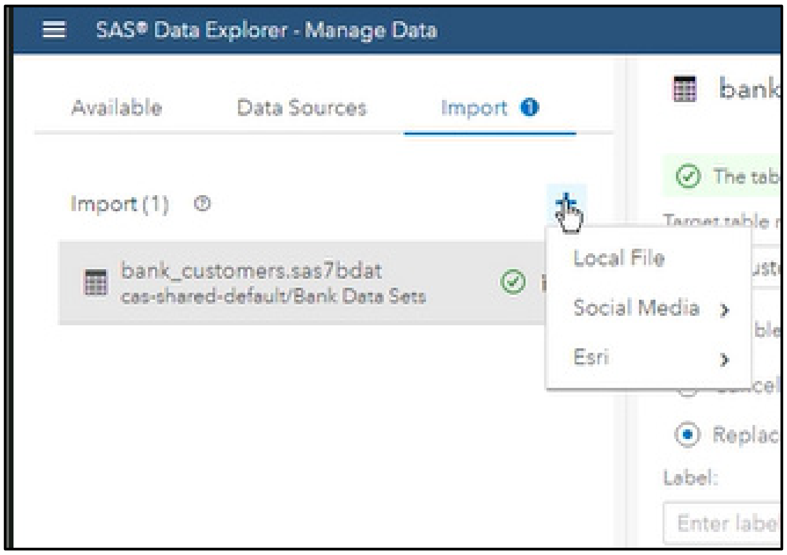
ZIP_POSTAL_CODE char
PHONE char
FAX char
EMAIL char
COMPANY char

After you have finished filtering, select "Import Item." The import loads the data into your SAS Viya environment in distributed format, and also loads the data into your Viya memory. After the import is complete, you can return to the Available tab on the left side of the wizard and you will see the table that you imported with a lightning bolt, which indicates it is an in-memory table.

Import Local Files

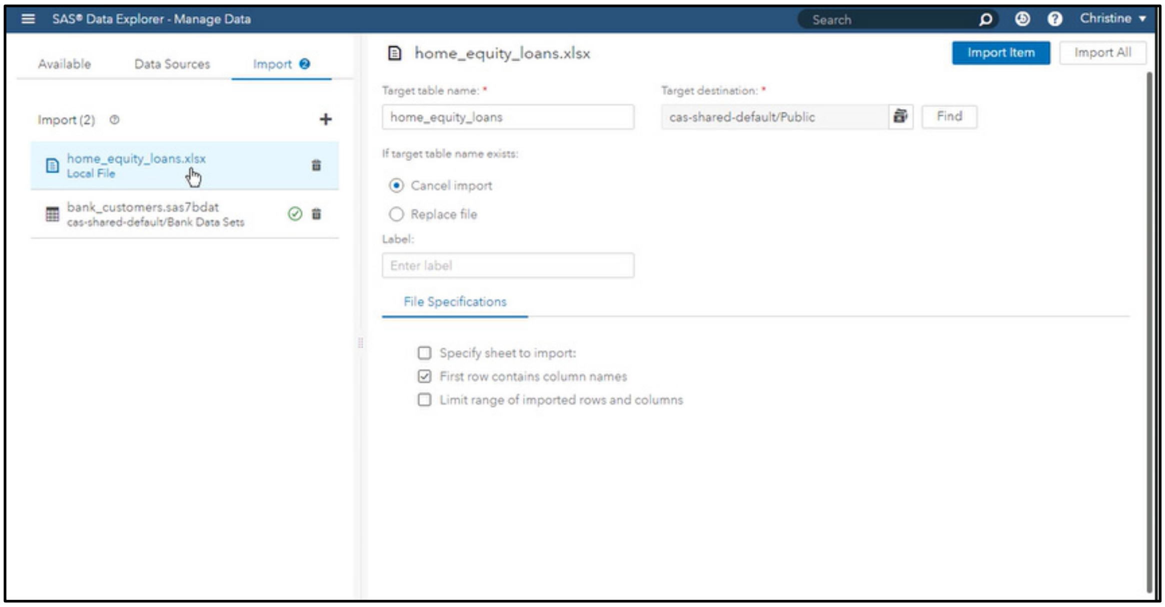
If you have a local file on your system that you want to import, there is another method to get that data into the SAS Viya environment. On the left side of the Choose Data window, select the Import tab. Click on the + sign, and choose "Local File" from the drop-down menu, as shown in Figure 4.9. A window will open that enables you to navigate to the location where the file, you want to import is located.

Figure 4.9: Import Local File



In this example, we are importing an Excel spreadsheet, so the Import tab will show certain file specifications that are available for Excel spreadsheets, as depicted in Figure 4.10. The options in the top half of the tab are the same as the previous example of importing a SAS data set, but you can also specify which sheets you want to import, indicate a header row, and limit the range of imported columns. When you have finished choosing all of your option, select "Import Item."

Figure 4.10: Import Tab



home_equity_loans.xlsx

Target table name: * home_equity_loans

Target destination: * cas-shared-default/Public

If target table name exists:

[✓ Cancel import] Replace file

Label: Enter label

- ☐ Specify sheet to import:
- ☑ First row contains column names
- ☐ Limit range of imported rows and columns

Perform Actions

After you import an item successfully, you will see a green notification at the top of the screen. If you click on "Action" in the notification, you will see options for other operations that you can do in your SAS Viya environment, including

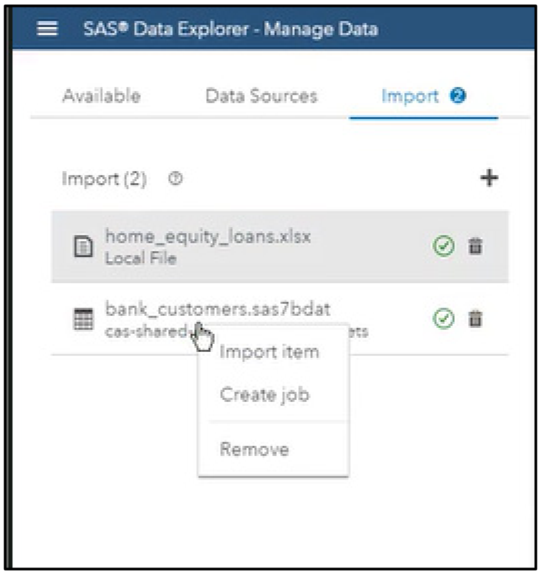
- Prepare Data – allows you to cleanse and wrangle data in SAS Data Studio
- Explore and Visualize Data – opens SAS Visual Analytics
- Build Models – lets you build models from the data
- Explore Lineage – opens SAS Lineage Viewer to explore object relationships

You can also access these features from the Actions drop-down menu in the upper right corner of your screen. From this one SAS Data Explorer application, you can see the interoperability in the SAS Viya Environment with applications that make seamless navigation possible. You will learn more about SAS Data Studio and SAS Lineage Viewer later in this chapter.

Create Jobs

In a previous section, we mentioned creating jobs that can be scheduled in a production environment. To create a job, go to the Import tab and right-click on any import job that you have created. Select "Create job" as shown in Figure 4.11.

Figure 4.11: Create Job Option



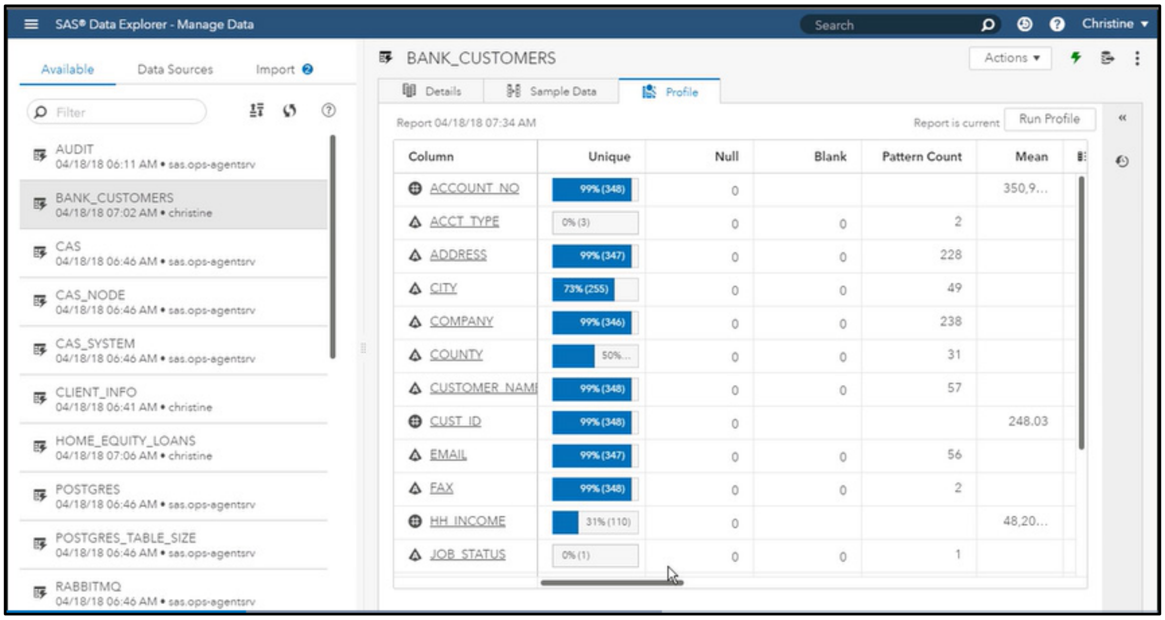
In the Create Job window that appears, you will see a default job name with a unique ID at the end. You have the ability to change this job name if you want and enter in a description. After you click OK to create the job, you can then go into the SAS Viya Environment Manager under the scheduling page and schedule that job to run based on specific triggers. Later on, you can go into SAS Job Monitor and check the status of those jobs as they run.

Profile Data

In this section, you will learn how to use SAS Data Explorer to profile data in a SAS Viya environment to determine data anomalies and inconsistencies. We will be using and reviewing related profile reports such as descriptive measures and frequency distributions.

In a previous section, you learned how to import data. For each table, on the Details tab you are able to see metadata about that table. On the Sample Data tab, you can see a sample of the first 100 rows. On the Profile tab, you can run a report to see the columns and generate table metrics about values that are Unique, Null, Blank Counts, Pattern Counts, and statistics including mean, median, mode, standard deviation, and standard error. See Figure 4.12.

Figure 4.12: Profile Tab

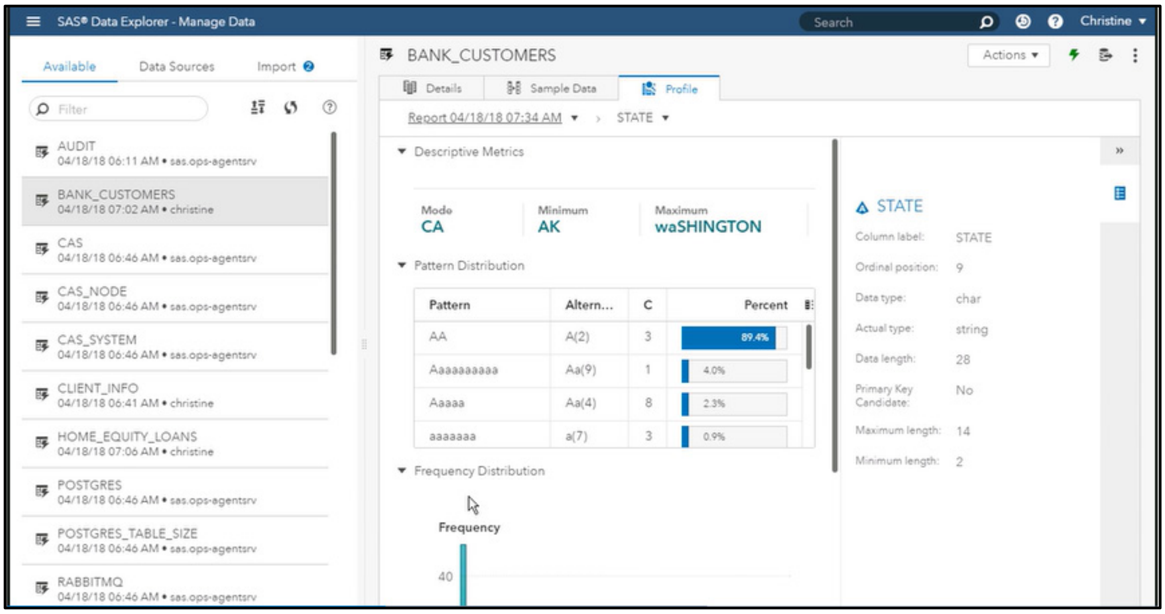

SAS® Data Explorer - Manage Data
Search
Christine
Available
Data Sources
Import
Filter
AUDIT
04/18/18 06:11 AM • sas.ops-agentsrv
BANK_CUSTOMERS
04/18/18 07:02 AM • christine
CAS
04/18/18 06:46 AM • sas.ops-agentsrv
CAS_NODE
04/18/18 06:46 AM • sas.ops-agentsrv
CAS_SYSTEM
04/18/18 06:46 AM • sas.ops-agentsrv
CLIENT_INFO
04/18/18 06:41 AM • christine
HOME_EQUITY_LOANS
04/18/18 07:06 AM • christine
POSTGRES
04/18/18 06:46 AM • sas.ops-agentsrv
POSTGRES_TABLE_SIZE
04/18/18 06:46 AM • sas.ops-agentsrv
RABBITMQ
04/18/18 06:46 AM • sas.ops-agentsrv
BANK_CUSTOMERS
Actions
Details
Sample Data
Profile
Report is current
Run Profile

Report 04/18/18 07:34 AM

Column	Unique	Null	Blank	Pattern Count	Mean
ACCOUNT_NO	99% (348)	0			350,9...
ACCT_TYPE	0% (3)	0	0	2	
ADDRESS	99% (347)	0	0	228	
CITY	73% (255)	0	0	49	
COMPANY	99% (346)	0	0	238	
COUNTY	50%...	0	0	31	
CUSTOMER_NAM	99% (348)	0	0	57	
CUST_ID	99% (348)	0			248.03
EMAIL	99% (347)	0	0	56	
FAX	99% (348)	0	0	2	
HH_INCOME	31% (110)	0			48,20...
JOB_STATUS	0% (1)	0	0	1	

If you want to review some of the data further, you can click on a column name in the Profile tab and bring up column descriptions with descriptive metrics including Mode, Min, and Max as shown in Figure 4.13. If you have redundancy in your data, then that may indicate issues. In the Pattern Distribution section, you can see the various patterns that currently exist in your data. The Frequency Distribution graph allows you to hover over the different bars to see certain values. On the far right-hand side, you will see some basic column information such as data type and data length. Clicking on the "Report" link at the top will take you back to the original Profile tab.

Figure 4.13: Column Description



AUDIT 04/18/18 06:11 AM • sas.ops-agentsrv
BANK_CUSTOMERS 04/18/18 07:02 AM • christine
CAS 04/18/18 06:46 AM • sas.ops-agentsrv
CAS_NODE 04/18/18 06:46 AM • sas.ops-agentsrv
CAS_SYSTEM 04/18/18 06:46 AM • sas.ops-agentsrv
CLIENT_INFO 04/18/18 06:41 AM • christine
HOME_EQUITY_LOANS 04/18/18 07:06 AM • christine
POSTGRES 04/18/18 06:46 AM • sas.ops-agentsrv
POSTGRES_TABLE_SIZE 04/18/18 06:46 AM • sas.ops-agentsrv
RABBITMQ 04/18/18 06:46 AM • sas.ops-agentsrv

Mode	Minimum	Maximum
CA	AK	waSHINGTON

Pattern	Altern...	C	Percent
AA	A(2)	3	89.4%
Aaaaaaaaaa	Aa(9)	1	4.0%
Aaaaa	Aa(4)	8	2.3%
aaaaaaa	a(7)	3	0.9%

STATE

Column label:	STATE
Ordinal position:	9
Data type:	char
Actual type:	string
Data length:	28
Primary Key Candidate:	No
Maximum length:	14
Minimum length:	2

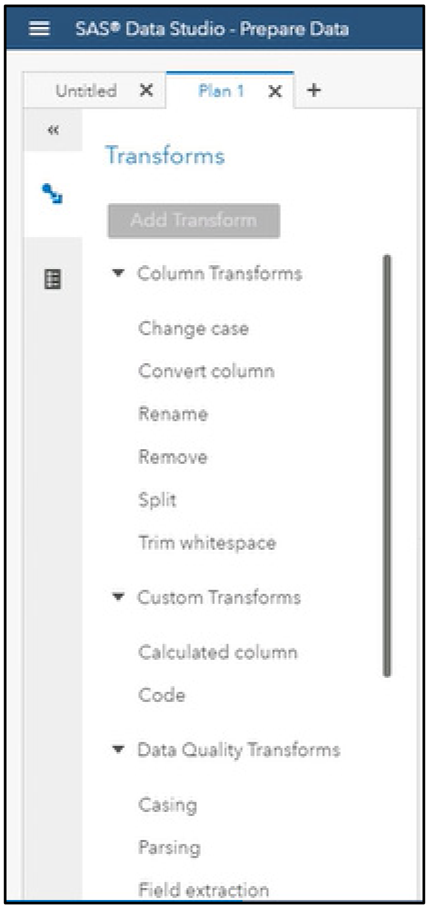
As you cleanse the data and create business rules, you may want to run the profile again with new business rules in place that cleans your data on the fly through machine learning or manual data cleansing processes. SAS Data Explorer gives you the ability to compare reports on the same table. Click on the circular arrow icon on the far right of the profile tab to view and compare versions of each report.

SAS Data Studio

In this section, you will learn how to use SAS Data Studio to prepare and cleanse data in your SAS Viya environment. In SAS Data Explorer, you have loaded tables into CAS memory or imported tables from a local file. When you review that data, you might notice some inconsistencies. In order to clean the data of anomalies and inconsistencies, select a table from the Available tab in Data Explorer. Click on the Actions drop-down menu in the upper right corner of your screen and select "Prepare Data." This will open up SAS Data Studio.

When you first enter SAS Data Studio, you have the option to create a New Plan or open an existing plan. After you create a New Plan and choose your data from available tables loaded in CAS memory, on the left side of the Data Studio window you will be presented with a list of Column Transforms, Custom Transforms for submitting custom code such as CASL or DATA steps, Data Quality Transforms, Multi-input Transforms, and Row Transforms. See Figure 4.14.

Figure 4.14: Transform Options



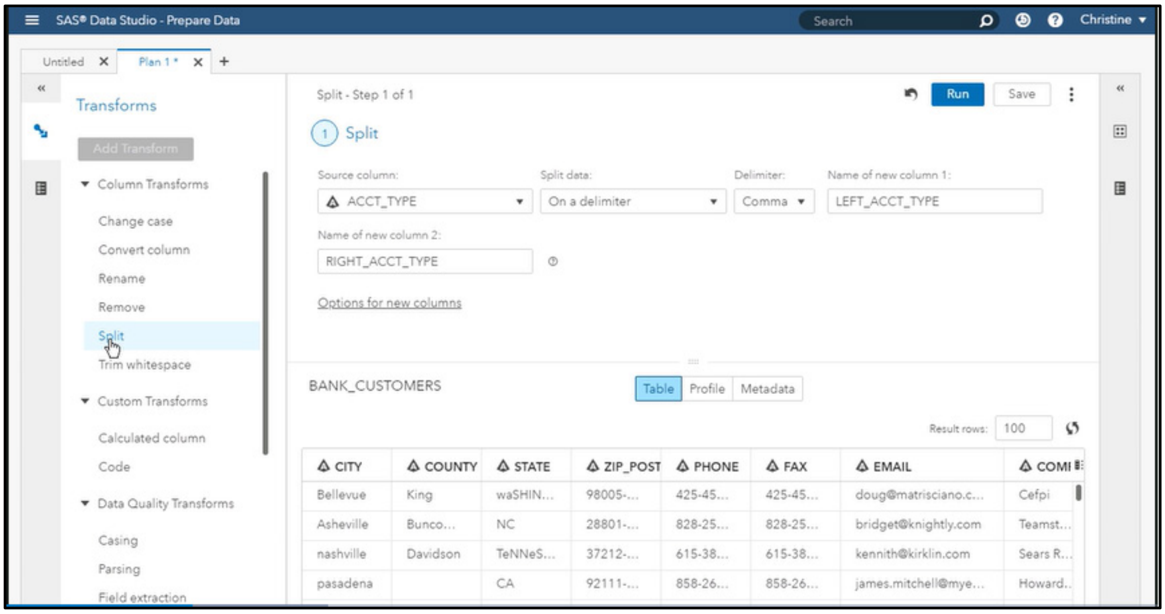
Column Transforms

Let’s look at an example of a column transform that can be performed in SAS Data Studio.

Split

If you have a table with a column that contains email addresses, you may want to break the email addresses into two separate fields: sender and domain. In order to do this, we will use the Split option under Column Transforms. Double-click “Split” to add the Split to your work palette, as shown in Figure 4.15.

Figure 4.15: Split



Split - Step 1 of 1

1 Split

Source column: ACCT_TYPE

Split data: On a delimiter

Delimiter: Comma

Name of new column 1: LEFT_ACCT_TYPE

Name of new column 2: RIGHT_ACCT_TYPE

BANK_CUSTOMERS

Result rows: 100

CITY	COUNTY	STATE	ZIP_POST	PHONE	FAX	EMAIL	COMI
Bellevue	King	waSHIN...	98005-...	425-45...	425-45...	doug@matrisciano.c...	Cefpi
Asheville	Bunco...	NC	28801-...	828-25...	828-25...	bridget@knightly.com	Teamst...
nashville	Davidson	TeNNeS...	37212-...	615-38...	615-38...	kennith@kirklin.com	Sears R...
pasadena		CA	92111-...	858-26...	858-26...	james.mitchell@mye...	Howard..

You can then select the column in the table that you want to perform the split on in the Source Column Field. You can split the data on a delimiter or other options in the Split data field. You can then choose the delimiter in the Delimiter field. In this example, we would choose the Source Column Email, split the data on a delimiter, and choose Other as the type of the delimiter, then specify the @ sign as the delimiter. The split will create two new columns, one for the left and one for the right. If you want to change the names of the new columns, click on the “Options for new columns” link where you can change the name, data type, length, or apply a label or SAS format.

When you have chosen all of your options, click “Run” in the upper right. You will see your changes applied in the table at the bottom of the screen. At this point, you have been wrangling the data in-memory without having to create a new table.

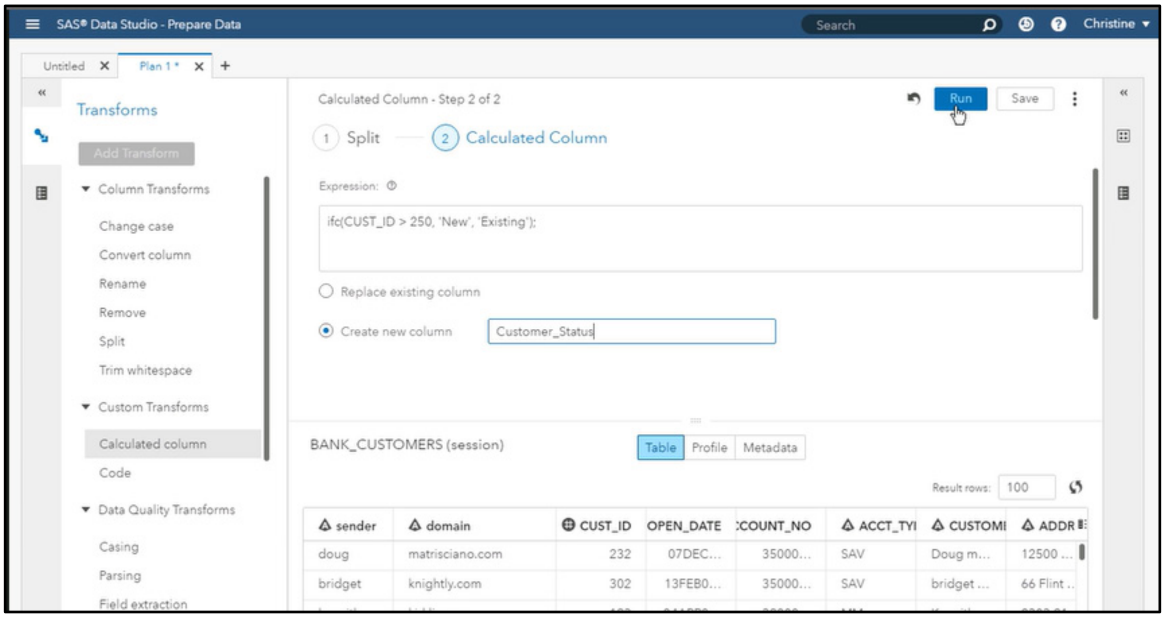
Custom Transforms

You can continue to transform your data by adding another step. In this section, we will look at the two types of custom transforms that are available in Data Studio.

Calculated Column

In this example, we will add another transformation by clicking on “Calculated column” and copying a SAS IFC function into the Expression box, as shown in Figure 4.16. The IFC function basically does a string comparison that says if CUST_ID is greater than 250, then I will create a new column.

Figure 4.16: Calculated Column


Calculated Column - Step 2 of 2

Expression:

```
ifc(CUST_ID > 250, 'New', 'Existing');
```

○ Replace existing column

● Create new column: Customer_Status

BANK_CUSTOMERS (session)

sender	domain	CUST_ID	OPEN_DATE	:COUNT_NO	ACCT_TYI	CUSTOMI	ADDR
doug	matrisciano.com	232	07DEC...	35000...	SAV	Doug m...	12500 ...
bridget	knightly.com	302	13FEB0...	35000...	SAV	bridget ...	66 Flint ..

We need to create a new column to house the new data, so select the “Create new column” option below the Expression field and re-name the column “Customer_Status”.

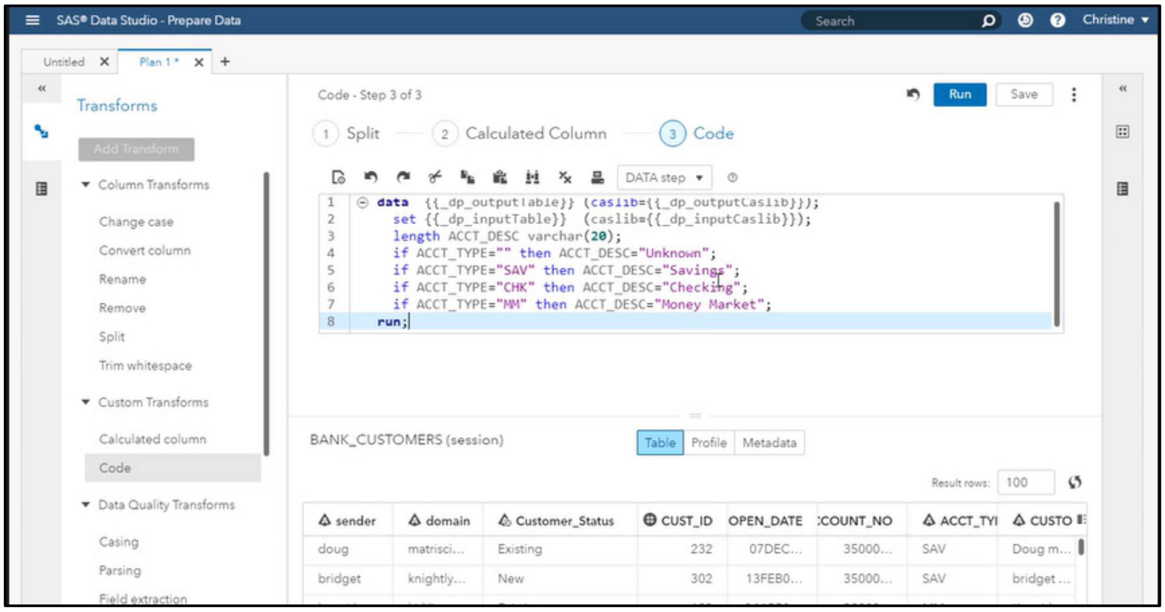
After you select Run, the Split Transform will run first, then the Calculated Column will run. You will see the new column added to the table at the bottom of the screen.

SAS Code

What if you have existing SAS DATA step code or new CAS language programming code (CASL) you want to apply? You can double click on Code in the Custom Transforms section. It adds another step that allows you to specify DATA step code or CASL to call CAS action sets, such as impute missing values.

In this example, we will use DATA step code by copying and pasting existing code into the editor window as shown in Figure 4.17. The code adds a new column called Account Description based on some criteria from ACCT_TYPE.

Figure 4.17: Code


Code - Step 3 of 3

```
data  {{_dp_outputTable}} (caslib={{_dp_outputCaslib}});
  set {{_dp_inputTable}}  (caslib={{_dp_inputCaslib}});
  length ACCT_DESC varchar(20);
  if ACCT_TYPE="" then ACCT_DESC="Unknown";
  if ACCT_TYPE="SAV" then ACCT_DESC="Savings";
  if ACCT_TYPE="CHK" then ACCT_DESC="Checking";
  if ACCT_TYPE="MM" then ACCT_DESC="Money Market";
run;
```

BANK_CUSTOMERS (session)

sender	domain	Customer_Status	CUST_ID	OPEN_DATE	:COUNT_NO	ACCT_TYI	CUSTO
doug	matrisci...	Existing	232	07DEC...	35000...	SAV	Doug m...
bridget	knightly...	New	302	13FEB0...	35000...	SAV	bridget ...

As a brief aside, if you look at the code in Program 4.1, you will see that everything from the length code down to the run is the same as SAS 9. What is different is the first two lines of code. In the first line, you specify the variables for the output table and your CAS library that is being referenced. In the second line, you specify the variables for the table you are reading in. These variables are needed because CAS works in a distributed environment and it creates intermediate tables. Data Studio will keep track of all of these intermediate tables for you.

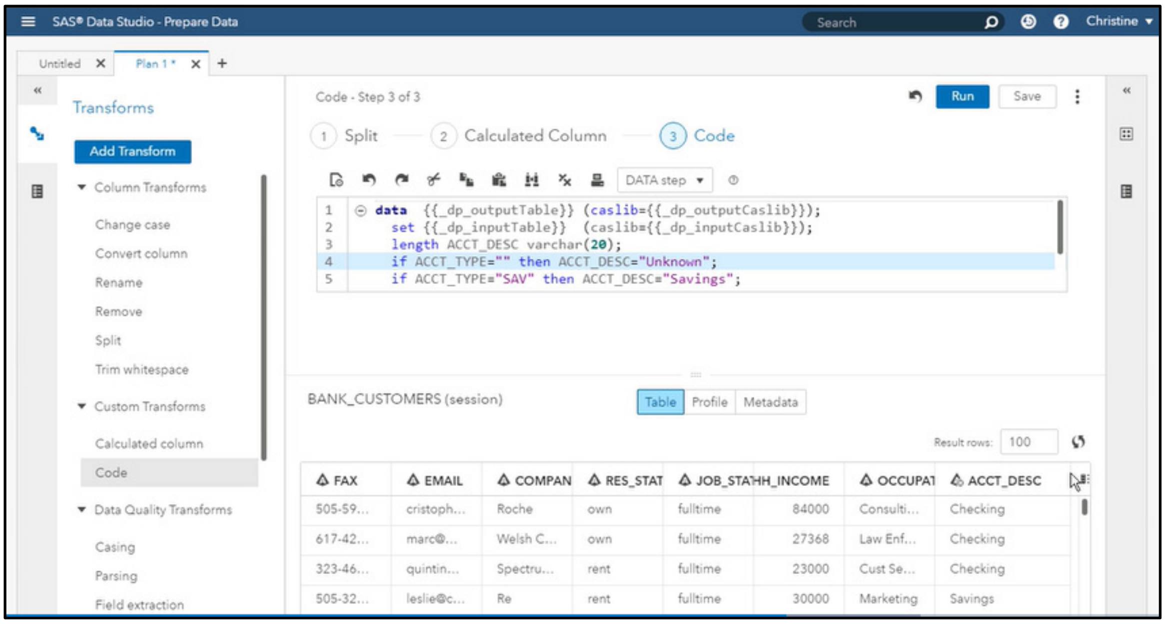
Program 4.1

```
data {{_dp_outputTable}} (caslib={{_dp_outputCaslib}});
    set {{_dp_inputTable}} (caslib={{_dp_inputCaslib}});
    length ACCT_DSC varchar(20);
    if ACCT_TYPE=”” then ACCT_DESC=”Unknown”;
    if ACCT_TYPE=”SAV” then ACCT_DESC=”Savings”;
    if ACCT_TYPE=”CHK” then ACCT_DESC=”Checking”;
    if ACCT_TYPE=”MM” then ACCT_DESC=”Money Market”;
run;
```

If you have existing DATA step code from SAS 9 that you want to leverage in Data Studio, just use the variables in place of what you are currently using in your existing code.

After you click run, you can review the results in the table at the bottom of the screen. As seen in Figure 4.18, a new column called ACCT_DESC was created based on the criteria specified in Program 4.1.

Figure 4.18: Code Results



```
data  {{_dp_outputTable}} (caslib={{_dp_outputCaslib}});
  set {{_dp_inputTable}}  (caslib={{_dp_inputCaslib}});
  length ACCT_DESC varchar(20);
  if ACCT_TYPE="" then ACCT_DESC="Unknown";
  if ACCT_TYPE="SAV" then ACCT_DESC="Savings";
```

FAX	EMAIL	COMPAN	RES_STAT	JOB_STA	HH_INCOME	OCCUPAT	ACCT_DESC
505-59...	cristoph...	Roche	own	fulltime	84000	Consulti...	Checking
617-42...	marc@...	Welsh C...	own	fulltime	27368	Law Enf...	Checking
323-46...	quintin...	Spectru...	rent	fulltime	23000	Cust Se...	Checking
505-32...	leslie@c...	Re	rent	fulltime	30000	Marketing	Savings

Data Quality Transforms

In this section, we will look at Data Quality Transforms.

Standardize

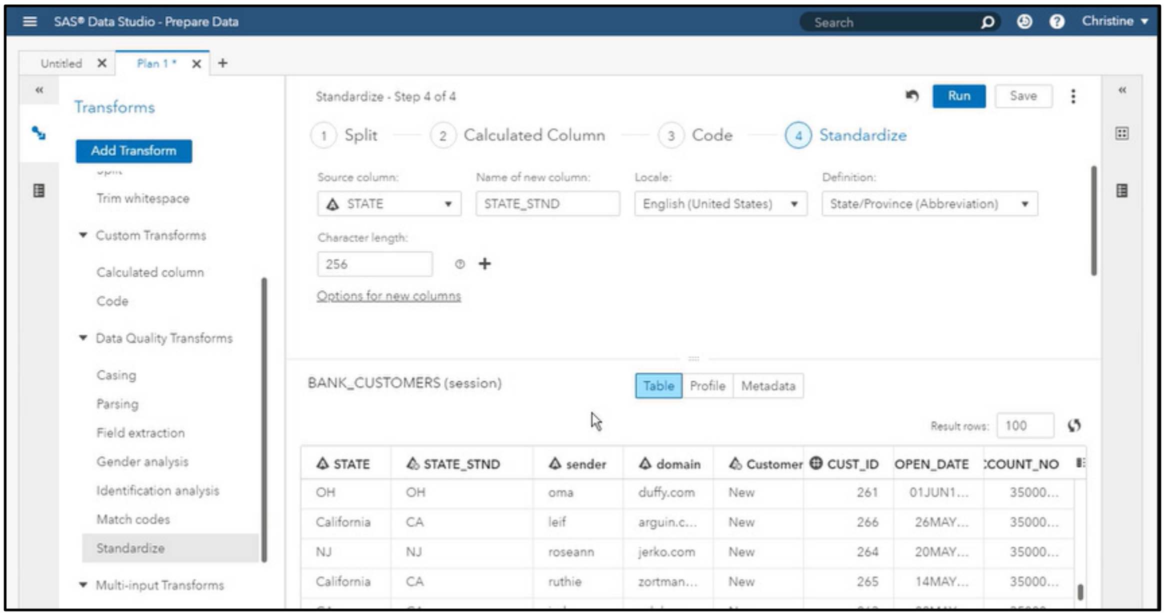
The Standardize data quality transform allows you to use the SAS Quality Knowledge Base to standardize your data into a consistent form. When you profiled your data earlier in SAS Data Explorer, you may have noticed that your data has some anomalies that need to be corrected. In this example, the data for State has both 2-byte codes and full state names.

Double-click Standardize in the Data Quality Transforms section. Then you can select your source column. The next field allows you to specify any name you want for your new column. In the next field, Locale, you can choose the locale for the SAS Quality Knowledge Base you want to use. The SAS Quality Knowledge base with its definitions and algorithms support over 40 different locales from around the world.

In the next field, Definitions, there are a lot of different definitions that are available for different data types. For this example, we will select State/Province (Abbreviation) because we want all of our State values to be abbreviated. You can also specify length and options for new columns, similar to the options in other transforms previously discussed.

After you click Run and review the results, you can see that in the new column, STATE_STND, the values for STATE have been standardized into the 2-byte code, as shown in Figure 4.19.

Figure 4.19: Standardize Results



Source column: STATE
Name of new column: STATE_STND
Locale: English (United States)
Definition: State/Province (Abbreviation)
Character length: 256

BANK_CUSTOMERS (session)

Result rows: 100

STATE	STATE_STND	sender	domain	Customer	CUST_ID	OPEN_DATE	:COUNT_NO
OH	OH	oma	duffy.com	New	261	01JUN1...	35000...
California	CA	leif	arguin.c...	New	266	26MAY...	35000...
NJ	NJ	roseann	jerko.com	New	264	20MAY...	35000...
California	CA	ruthie	zortman...	New	265	14MAY...	35000...

Also in Data Studio, you can look at the Profile and Metadata column information about this table and about the new fields that you have created.

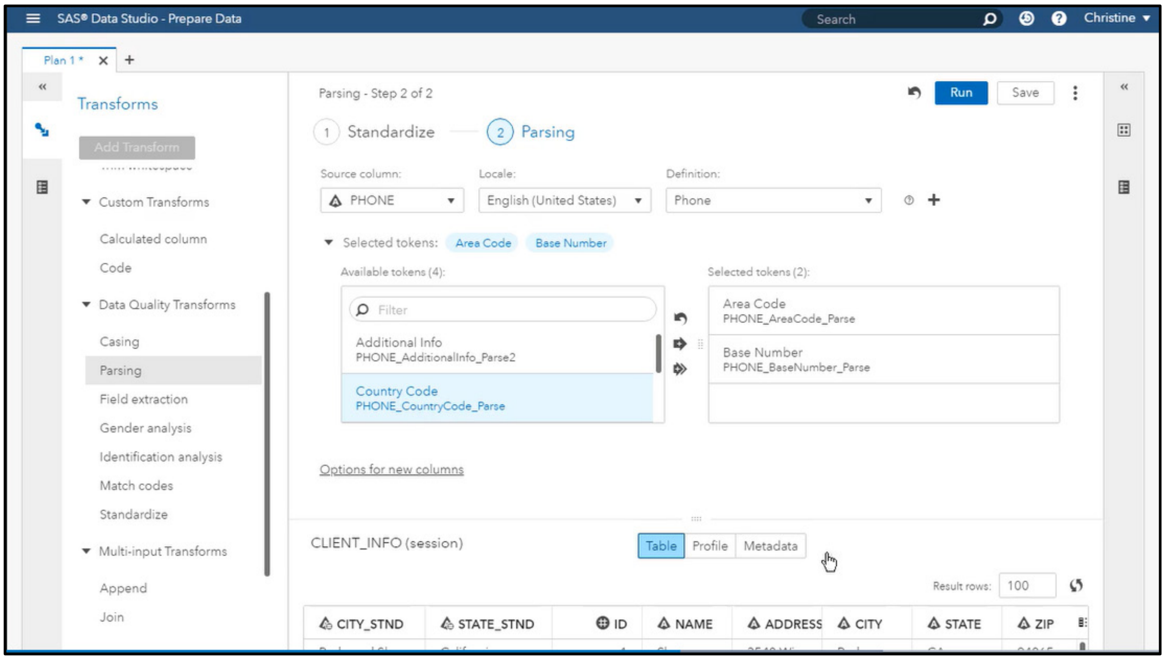
Parsing

Sometimes you want to break data down into its various semantic components or "parse" it so that you can analyze the individual components. For example, you may want to only analyze the ZIP codes from a full address or only analyze the year from a DD/MM/YY date.

The Parse transform is similar to the Standardize transform because it also uses the SAS Quality Knowledge base to extract information. In this example, we will parse data in a column called PHONE. Double click on Parsing in the Data Quality Transforms section. Select the column name from the Source Column field, as well as the Locale you want to use. In the definition, we will select Phone because we are parsing a phone number.

Next, you will see the individual tokens into which your data type can be parsed. For a phone number, we will select Area Code and Base Number from the list of tokens, as shown in Figure 4.20. This will break the data down into two fields.

Figure 4.20: Parsing



Parsing - Step 2 of 2

1 Standardize — 2 Parsing

Source column: PHONE
Locale: English (United States)
Definition: Phone

Selected tokens: Area Code, Base Number

Available tokens (4):
- Additional Info — PHONE_AdditionalInfo_Parse2
- Country Code — PHONE_CountryCode_Parse

Selected tokens (2):
- Area Code — PHONE_AreaCode_Parse
- Base Number — PHONE_BaseNumber_Parse

Options for new columns

CLIENT_INFO (session)

Result rows: 100

CITY_STND	STATE_STND	ID	NAME	ADDRESS	CITY	STATE	ZIP

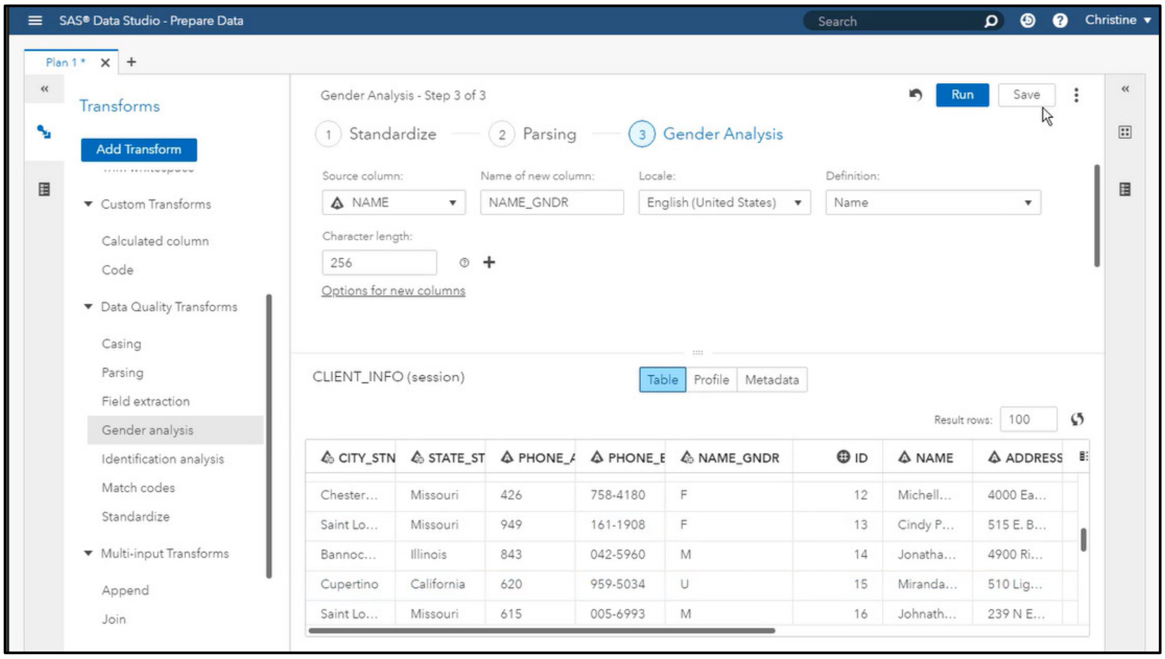
After you click Run, you will see two new columns titled PHONE_AreaCode and PHONE_BaseNumber that contain the parsed information.

Gender Analysis

Let's look at one final example of a data quality transform. The Gender Analysis transform allows you to know the gender associated with the name of an individual based on the information in the SAS Quality Knowledge Base. The algorithm will determine whether a data value is male, female, or unknown. This may be useful in marketing or clinical trial scenarios.

Double-click on Gender Analysis in the Data Quality Transform section. Select the column name from the Source Column field, choose the name of your new column, and choose the Locale you want to use. In the definition, we will select Name. Select Run to complete the transform. As you can see in Figure 4.21, a new column is added with F, M, and U values based on the names in the NAME column.

Figure 4.21: Gender Analysis Results



Gender Analysis - Step 3 of 3

1 Standardize — 2 Parsing — 3 Gender Analysis

Source column: NAME
Name of new column: NAME_GNDR
Locale: English (United States)
Definition: Name
Character length: 256

Options for new columns

CLIENT_INFO (session)

Result rows: 100

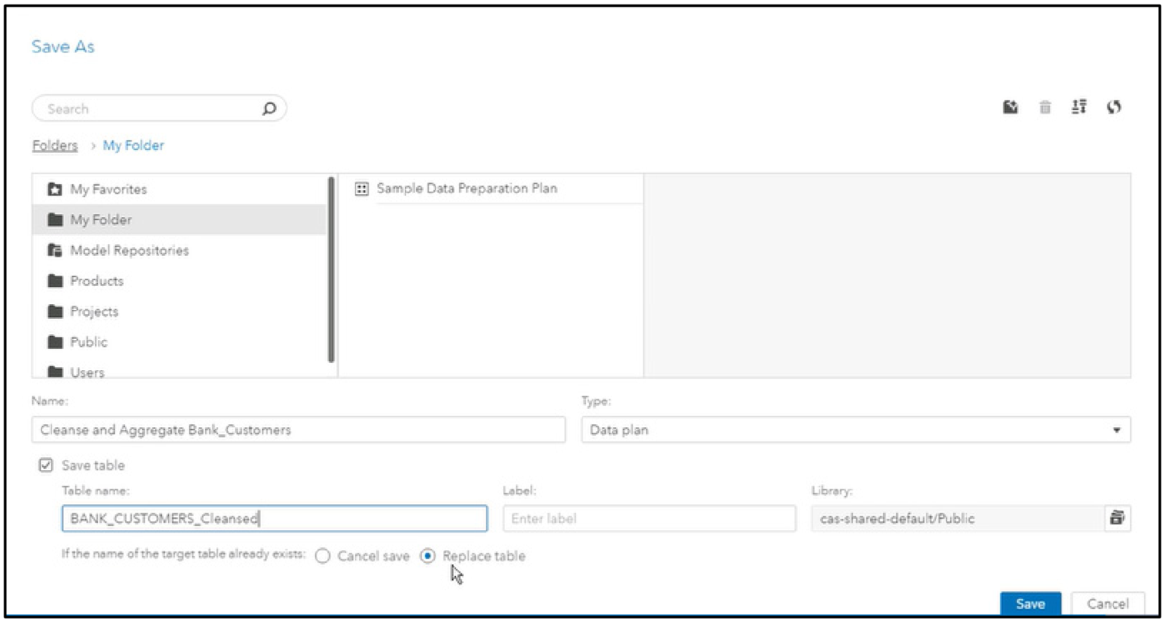
CITY_STN	STATE_ST	PHONE_A	PHONE_E	NAME_GNDR	ID	NAME	ADDRESS
Chester...	Missouri	426	758-4180	F	12	Michell...	4000 Ea...
Saint Lo...	Missouri	949	161-1908	F	13	Cindy P...	515 E. B...
Bannoc...	Illinois	843	042-5960	M	14	Jonatha...	4900 Ri...
Cupertino	California	620	959-5034	U	15	Miranda...	510 Lig...
Saint Lo...	Missouri	615	005-6993	M	16	Johnath...	239 N E...

Saving Results

Once you have completed all of your transforms and your data is how you expect it to be, you can save this data preparation plan and the results to an actual table.

Click on the Save button in the top right next to the Run button. You can give your data plan a title and also save the table with a new name or replace an existing table by using the options as shown in Figure 4.22. You can also specify where you want to save the actual CAS table.

Figure 4.22: Save As Window



Name: Cleanse and Aggregate Bank_Customers
Type: Data plan
[✓] Save table
Table name: BANK_CUSTOMERS_Cleansed
Label: Enter label
Library: cas-shared-default/Public
If the name of the target table already exists: Cancel save [✓ Replace table]

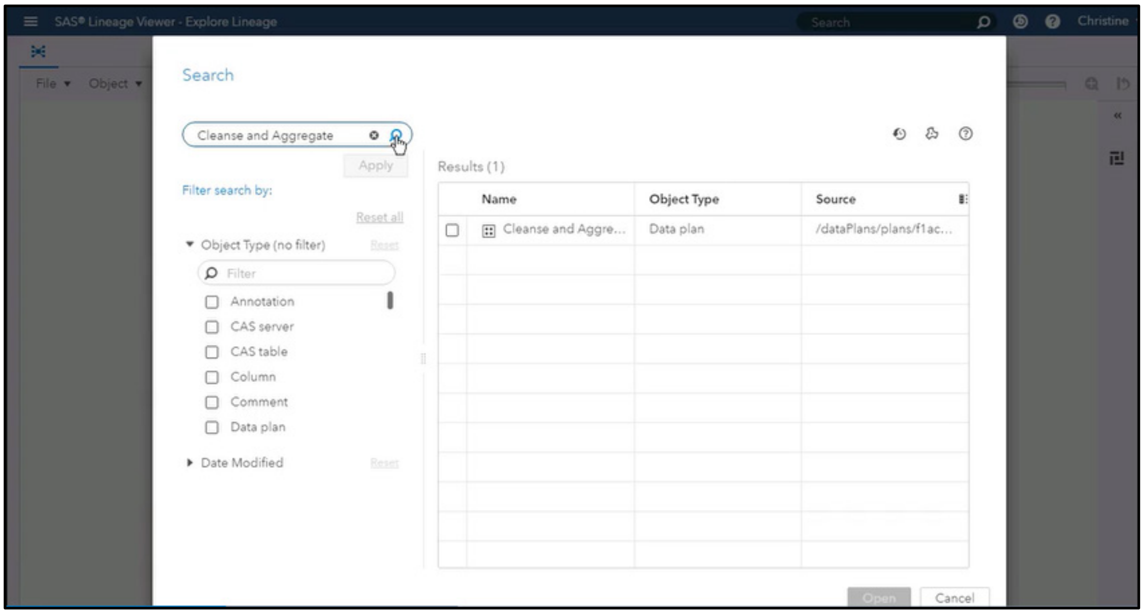
If you want to save and then create a job from this data preparation plan, you can click on the three vertical dots next to the Save button and select "Create job" from the drop-down menu. This will enable you to create a reusable job that you can run through the scheduling interface and environment manager.

SAS Lineage Viewer

Once you have ingested your data into your SAS Viya environment using SAS Data Explorer and created data preparation plans using SAS Data Studio, you will want to be able to manage and govern those assets. To help you visualize and perform those processes, you can use SAS Lineage Viewer.

The first step to explore lineage is to open up a Search for subjects by clicking on File in the upper left corner of the Lineage Viewer screen and selecting "Search for subjects." This will enable you to search the lineage repository for data objects and assets. In a previous section, we created a data preparation plan called "Cleanse and Aggregate," so in this example, we will search for that data preparation plan as shown in Figure 4.23 by typing into the search box and clicking the magnifying glass search button.

Figure 4.23: SAS Lineage Viewer Search



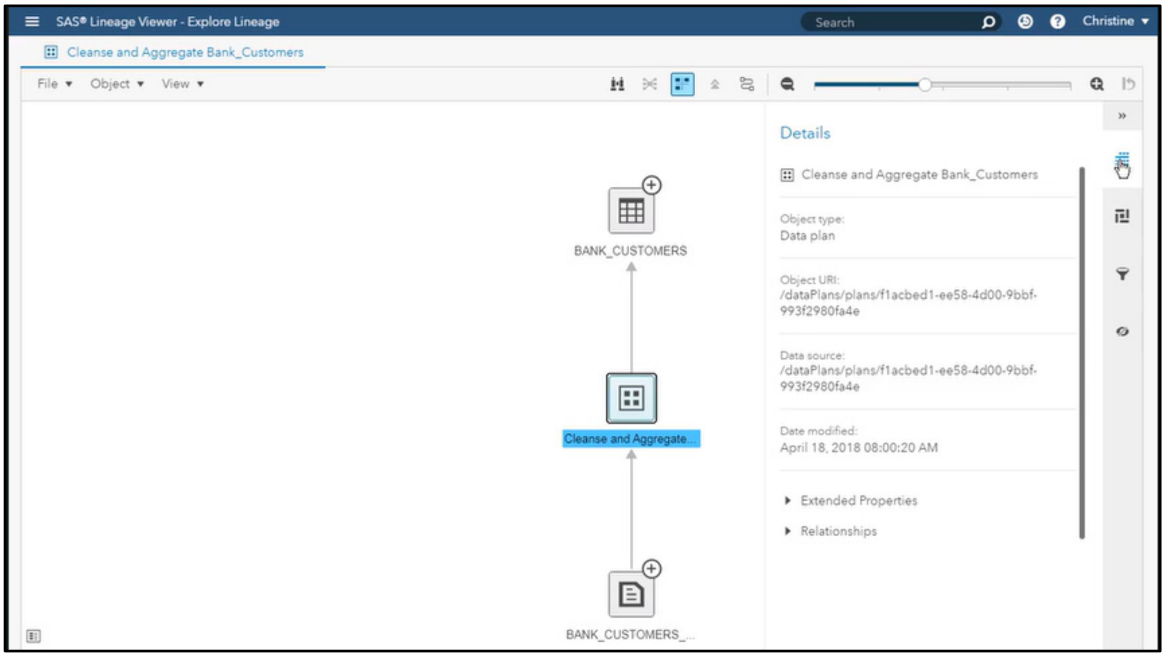
Name	Object Type	Source
Cleanse and Aggre...	Data plan	/dataPlans/plans/f1ac...

The search results will show you any data objects such as data preparation plans, tables, data sources, schemas, and files that meet your search criteria. Clicking on a results entry will open the visual environment to enable you to view relationships, govern the data, and understand what happens if you remove a table.

Details Tab

In the visual environment, you can click on an object and then open the details tab on the right side of your screen. The Details tab tells you what the object type is, the URI, the data source, when it was modified, as well as the relationships of that object to other tables and files, as shown in Figure 4.24.

Figure 4.24: Details Tab



Cleanse and Aggregate Bank_Customers

Object type:
Data plan

Object URI:
/dataPlans/plans/f1acbed1-ee58-4d00-9bbf-993f2980fa4e

Data source:
/dataPlans/plans/f1acbed1-ee58-4d00-9bbf-993f2980fa4e

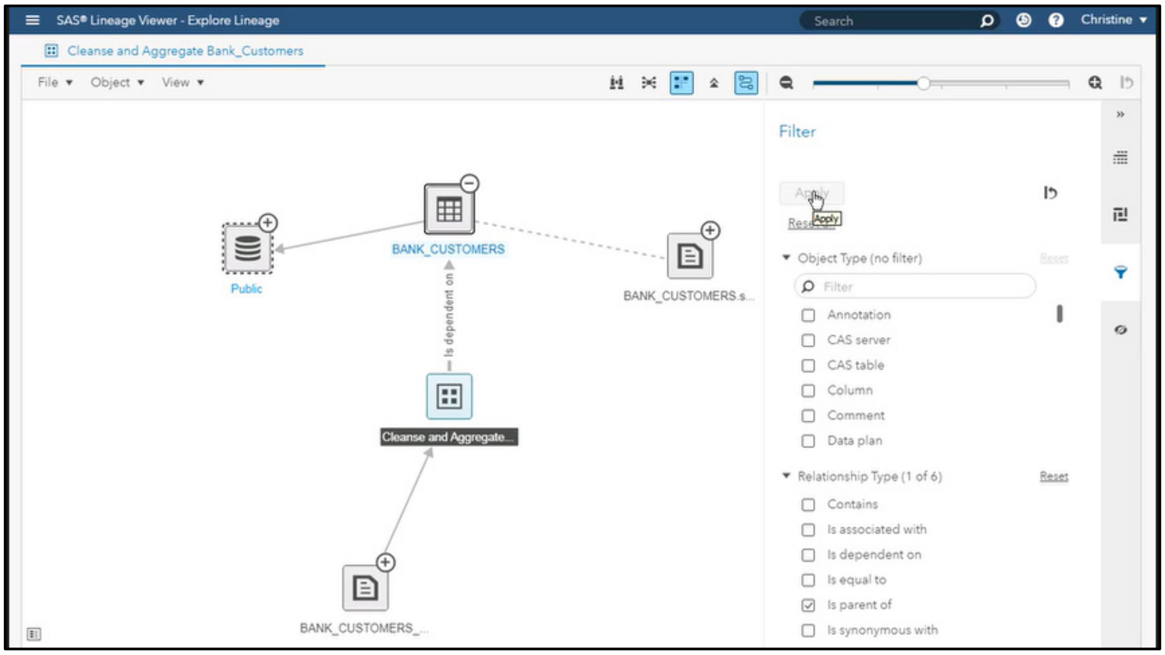
Date modified:
April 18, 2018 08:00:20 AM

Another way to view the relationships between objects is to click on the + sign in the upper right corner of an object. This expands the relationships in the visual interface. If you want even more information about the relationships between objects, you can select the “Highlight Path” icon at the top of the screen.

Manage View Filters Tab

If you need to filter your data, you can select the filter icon from the right side of the screen. (See Figure 4.25.) This allows you to remove certain objects or certain paths from this view. You can Reset to remove all filters and return to the original view.

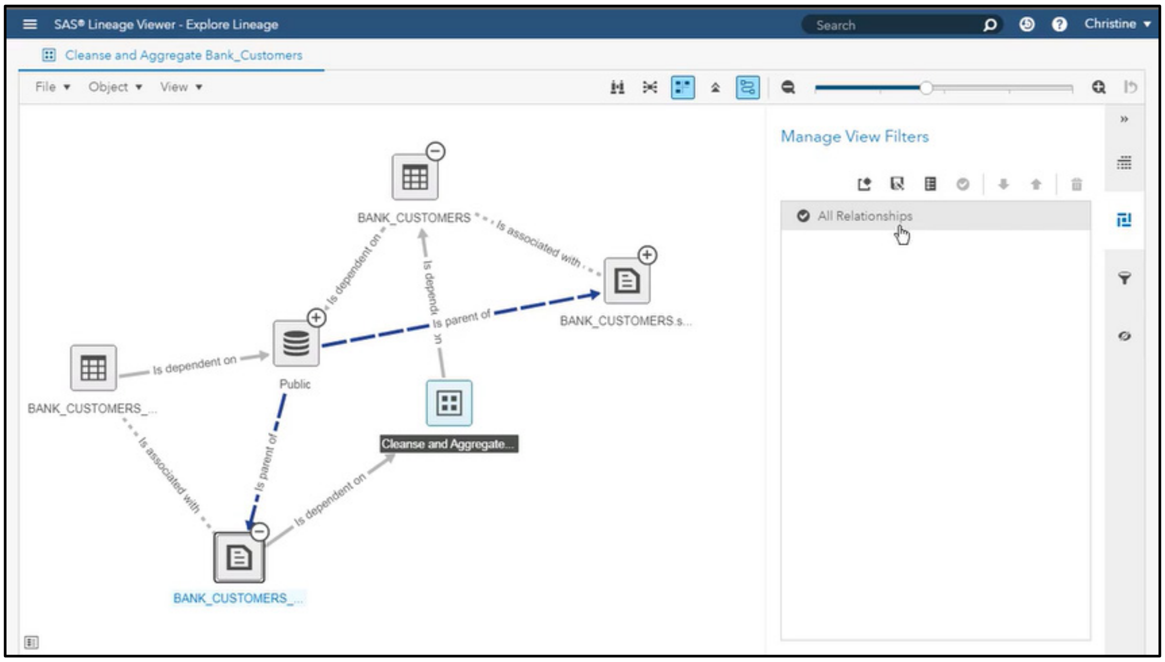
Figure 4.25: Filter



Manage View Filters

When you first enter Lineage Viewer, you will see the Default view that shows all relationships between data objects, as seen in Figure 4.26.

Figure 4.26: Default View



You can create specialized views by clicking on the New button, which opens a dialog to create a new view based on the relationship types and object types that you want to be displayed in the view. After you create a new view, you can use the checkmarks in the Manage View Filters tab to change the custom view to be the default view.

If you have a specific object that you want to see only the relationships based on that data, you can select that object and click the "View Lineage" icon at the top of the screen. This will create a new view just for that specific object. An easy way to reset the view is to click on the View drop-down menu in the upper left of the screen and choose "Reset layout."

Resources

This chapter is based on the "SAS Data Management" videos in "SAS® Viya® Enablement," a free course available from SAS Education.

You may find the following documentation helpful as you learn more about data management in SAS® Viya®:

- *SAS® Cloud Analytic Services 3.2: Fundamentals*
- *SAS® Viya® 3.4: Data Preparation*
  - SAS® Data Explorer 2.2: User's Guide
  - SAS® Data Studio 2.2: User's Guide
  - SAS® Lineage Viewer 2.2: User's Guide

Access free tutorials and other data preparation training resources from the SAS Data Preparation Learning Center.

For more information about SAS® Viya®, visit https://www.sas.com/en_us/software/viya.html.

For more information about data management, visit sas.com/data-preparation.