SAS® GLOBAL FORUM 2020 #SASGF

Paper 5109-2020

Using a Microsimulation to Evaluate Population and Policy Changes on Program Integrity Priorities

Jonathan Smith, Centers for Medicare and Medicaid; Meghan Beckowski, Deloitte Consulting LLP; Michael Greene, Deloitte Consulting LLP; Andrew Kemple, Emendata, LLC; Bil Westerfield, A. Reddix & Associates Inc. (ARDX) and Magpie Health Analytics

ABSTRACT

The Centers for Medicare & Medicaid Services (CMS) conducts Risk Adjustment Data Validation (RADV) to ensure the accuracy and integrity of data submitted for Medicare Advantage (MA) payments. CMS requires a simulation to forecast future program integrity trends. The RADV Synthetic Plan Microsimulation makes individual-level projections with a series of machine learning models and uses an agent-based simulation framework to forecast outcomes. SAS Viya and Python are used together in a streamlined framework, from data manipulation, to modeling, to simulation, and finally to visualization. Beneficiary data, including social determinants of health, are simulated with machine learning models such as enrollment events, health conditions, and payment error. Bootstrapping enables CMS to estimate variability in future payment error via point estimates and confidence intervals enabling CMS to compare various sampling strategies. A combination of code optimization and work parallelization in SAS Viya deployed in a cloud environment improves and distributes the required computation, reducing the run time of the bootstrapping process for multiple sampling strategies from nearly a year to only a few hours. Displaying the results of the sample strategies through an interactive dashboard in SAS Visual Analytics provides CMS with high-level information to help future decision making. The dynamic interactions and filters in this dashboard are comprehensive and user-friendly. Stakeholders can make policy and program integrity decisions with this interactive, scenario-based approach.

INTRODUCTION

Within the Centers for Medicare & Medicaid Services (CMS), the Center for Program Integrity (CPI) serves to protect the Medicare and Medicaid Programs from fraud, waste, and abuse. To help support this mission, its Risk Adjustment Data Validation (RADV) team is tasked with ensuring the accuracy and integrity of data submitted for Medicare Advantage payments. In its role overseeing these efforts, CMS sought a means of studying how different scenarios and policy actions can affect the Medicare Advantage environment.

To deliver this capacity to CMS, the RADV Synthetic Plan Microsimulation was created; the microsimulation makes individual-level projections multiple years in the future using beneficiary data and machine learning models. To estimate the effects of policy changes and trends on Medicare Advantage payment accuracy, the microsimulation is run under various scenarios using parameters within the model. Its functionality can be used to alter assumptions at each major point in the process of validating Medicare Advantage payments, providing an opportunity for CMS to examine a wide variety of "what-if" scenarios sometimes referred to as sensitivity analysis or scenario planning.

A combination of SAS Viya, SAS 9.4, and open source software such as R and Python provided a platform that could support the complex modeling framework needed to satisfy these expansive requirements. From the first step of data management through the final step of displaying the output, the microsimulation was implemented in SAS Viya, giving CMS access to three powerful sets of tools:

(1) *Leveraging Big Data Processing*—To create an accurate forecast, the microsimulation ingests a vast amount of data from both public and private sources. For handling these large data sizes, the microsimulation turns to SAS's Cloud Analytic Services (CAS) within the SAS Viya environment.

(2) *Integrating with Open-Source Software*—The microsimulation relies on advanced statistical algorithms to perform its predictions, some of which are available in SAS and others through open-source software. To deploy open-source machine learning models that can augment SAS tools, the microsimulation integrates Python and R code within SAS Viya.

(3) *Using Interactive Visualizations*—The microsimulation's business users need ready access to its results. SAS Visual Analytics offers a means of creating visualizations within SAS Viya that let users dynamically create what-if scenarios to assist in future decision making.

This paper is organized to share key ideas that were useful for standing up this complex system in SAS Viya. After an introduction of the microsimulation framework, the remaining sections discuss the three categories of tools in turn. They explain why the microsimulation needed these capacities and outline a few main takeaways with supporting examples and references.

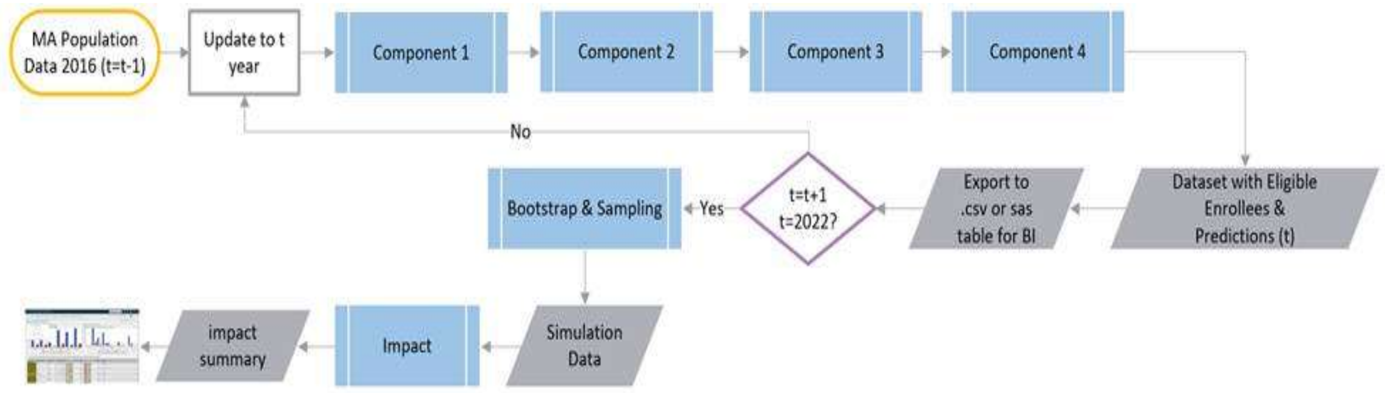
SIMULATION FRAMEWORK

The RADV Synthetic Plan Microsimulation framework was implemented as a series of SAS and Python programs. Each program or component of the framework represents either a data processing step or a statistical or machine learning model. These components are linked together in a logical flow intended to simulate the progression of Medicare Advantage beneficiaries over time, concluding with the potential for being sampled to help verify the accuracy of payments made by CMS. Many of the components of the framework are parameterized. This enables users to run scenarios for different policy options and compare the resulting Key Performance Indicators (KPIs) in SAS visualizations. As data volumes are quite large, with some datasets including hundreds of millions of observations, the Synthetic Plan Microsimulation leverages SAS CAS functionality wherever possible. Use of these tools are discussed in the following section on leveraging big data processing.

A microsimulation is a technique where observations of individual members in a population are used to simulate the outcomes of a process or government program such as Medicare Advantage. The individual data are used to forecast future behavior or events and aggregated to the whole population or subgroups (Rutter 2011; Baker & Thompson 2017; Lomax & Smith 2017; Marois & Belanger 2014). These types of agent-based microsimulation models have become increasingly used to forecast scenarios under uncertainty (Lewin Group 2009; Blumberg 2003; Glied & Tilipman 2010). When these observations are forecast for a single period of time, the microsimulation is considered a static simulation, when forecasts are made over multiple periods of time, the microsimulation is considered a dynamic simulation (Mills). The RADV Synthetic Plan Microsimulation was developed as a dynamic simulation, with each time period representing a year where users are able to forecast potential scenarios multiple years in the future.

The diagram in Figure 1 below illustrates the framework of the Synthetic Plan Microsimulation. This pipeline of programs is fed with a dataset of Medicare beneficiary information at the individual level. Each step in the pipeline represents a forecast for a particular outcome or group of outcomes for the next year, such as health outcomes, mortality, enrollment in a Medicare Advantage plan or Traditional Medicare, etc. These forecasts are produced with probabilistic statistical or machine learning models created in SAS or with open source Python libraries.

Figure 1: Diagram of Synthetic Plan Pipeline



The input dataset is carried forward through each component, with outcomes from the calculation appended as columns on the dataset. The current version of the Synthetic Plan Pipeline generates forecasts for five years. After SAS processes each component in the pipeline and reaches the end of the loop, it outputs the final dataset for that year's prediction, and takes a copy of the dataset to start through the next loop of the pipeline. At the end of the pipeline, there are two additional components: bootstrap and visualization.

In order to estimate the variability in the forecasts, the Synthetic Plan Microsimulation framework includes a bootstrap resampling step. Bootstrap resampling is a non-parametric statistical methodology for estimating variance in KPIs based on sampling with replacement from the population of interest (Efron 1982, 1987; Barton & Schruben 1993). Bootstrap resampling was chosen to measure the variability instead of other methods such as analytical derivation of the variance or normal distribution approximation because bootstrapping does not require as stringent assumptions about the probabilistic distributions of outputs of the statistical and machine learning models. In particular, bootstrap resampling does not rely on assumptions of normality (Efron 1982), making this technique especially attractive for a complex assembly of machine learning predictions where distributional assumptions are not easily estimated. Bootstrap estimates are aggregated into a smaller dataset, organized by filters which is fed into the visualization component.

The visualization component provides illustrations of the metrics and estimates captured by the models in SAS Visual Analytics software which will be described in more detail in the Interactive Results section. Results from each model and the bootstrapping processes are aggregated from individual person-level results to various sub-populations (e.g., county/state demographic groups, CMS policy types, etc.) to match filtering criteria on the visualizations. The aggregated SAS datasets feed seven different visualizations to help users interpret results.

LEVERAGING BIG DATA PROCESSING

Creating a dynamic simulation for this application was a challenging computational problem that proved to be an excellent use case for leveraging CAS in SAS Viya. The microsimulation ingests a large amount of data as inputs (tens of millions of records) and, in the process of performing the simulation, produces a large amount of data (with some datasets on the order of hundreds of millions of records). Not only is the microsimulation working with "Big Data", it also faces intensive processing demands. The last step in the pipeline, bootstrap resampling of Medicare Advantage beneficiaries, results in hundreds of thousands of individual computations that SAS needs to execute.

This pipeline was originally developed in Base SAS 9.4, so once moved to SAS Viya, the code ran single-threaded in the foundational SAS environment for executing classic SAS program code: SAS Program Runtime Environment (SPRE). As the design of the microsimulation expanded, running on a single thread became more difficult. The challenging requirements drove towards using CAS to handle the large workload and data size in a scalable cloud environment. What follows are a few of the lessons learned in moving from single-threaded processing to CAS's multi-threaded processing of in-memory distributed data[1].

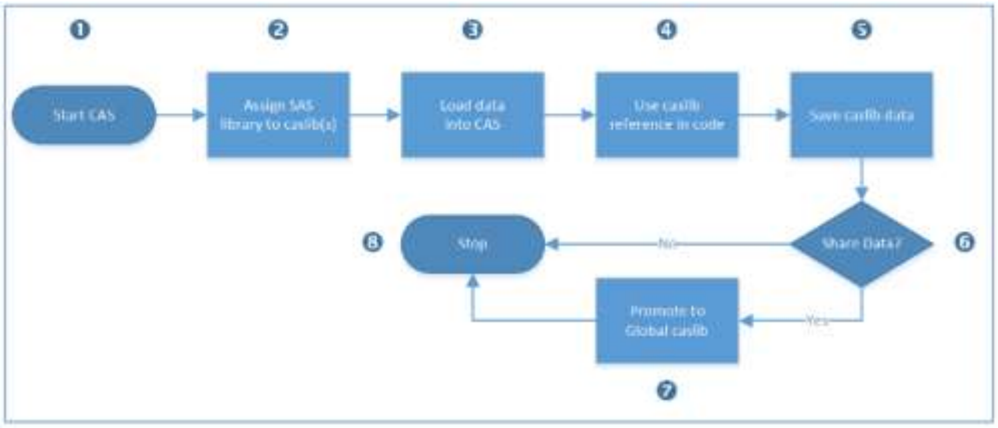
SHARING DATA IN CAS

Before getting started in CAS, the data needs to be loaded into memory. In cases where more than one person needs to work with the same dataset, it helps to maintain a persistent, centrally accessible dataset (Pendergrass 2017), which CAS facilitates through the use of a global CAS library.

The need to share data across several parties emerged on the Synthetic Plan Microsimulation, a collaboration that brought together multiple contributors from several organizations. To develop their part of the pipeline, each collaborator used the Medicare Advantage population described above, which needed to be loaded into memory before leveraging CAS. However, loading a large dataset from SPRE into memory incurs a computational cost[2]; without planning, this cost would be borne individually by each team member working with the Medicare Advantage population data.

The total runtime costs were able to be reduced by loading a dataset into memory that is persistent and accessible across team members. Using PROC CASUTIL, the data can be loaded into memory and promoted to a global CAS library. With this step, the up-front cost of loading the data into memory is incurred only once and everyone on the team receives access. Figure 2 shows the process by which data is loaded into CAS and highlights the point at which data can be promoted to a global CAS library (Shannon 2018).

Figure 2: Loading Data into CAS



(Source: Shannon 2018)

[1] For more information on SPRE and CAS, see the following post: https://communities.sas.com/t5/SAS-Communities-Library/Deploying-the-SPRE-in-SAS-Viya-3-4/ta-p/602891

[2] Benchmarking tests for the Synthetic Plan Microsimulation measured that loading data into memory from a CAS library took about a third of the time that it took to load data into memory from SPRE.

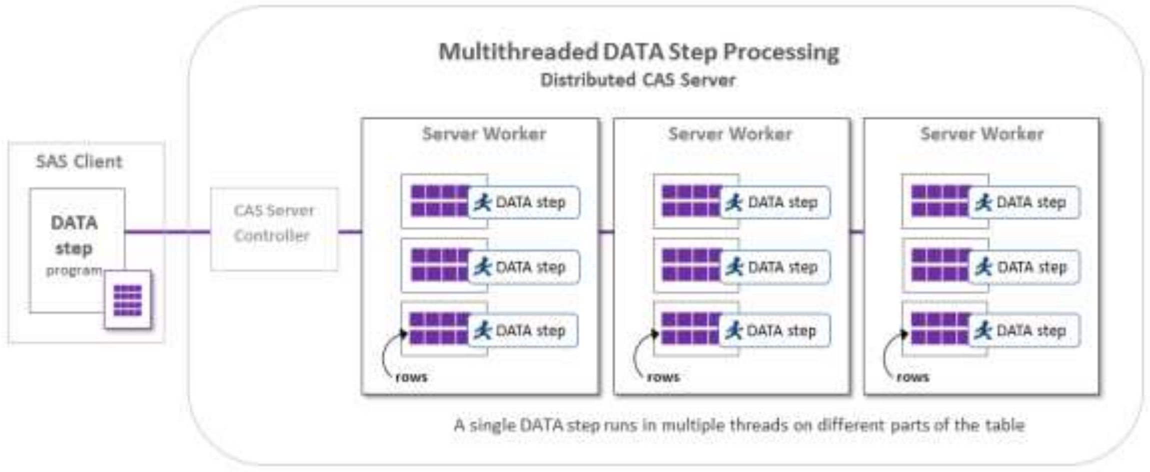
UNDERSTANDING ARCHITECTURE WHEN STARTING TO CODE IN CAS

After getting the mechanics of loading data settled, the next step was to make sure the code took full advantage of CAS functionality. This process should be treated as more than a substitution of syntax; it calls for thinking about what it means to work with in-memory distributed data.

This approach is particularly important for code that relies on DATA steps. For those using PROC steps in SAS Viya, SAS has made sure that several important procedures are CAS-enabled (SAS 2019a), making it easier to use the syntax "out of the box" (Sober 2018). For example, the pipeline components that fit and score models for the microsimulation use PROC steps that readily take advantage of the CAS architecture. However, the end of the pipeline relies heavily on DATA steps. For this kind of code, the transition to CAS rests on understanding the underlying architecture of CAS (Weiss 2018; SAS 2019c; SAS 2019d).

When starting to code in CAS, one fundamental change is to recognize that when a dataset is loaded into memory, CAS distributes its contents across multiple computers, or nodes (Secosky 2017; Weiss 2018). When a DATA step is encountered in the code, the threads on these nodes can be put to work; as Figure 3 shows (SAS 2019c), CAS has each thread simultaneously execute the logic of the DATA step on its allocated segment of the dataset. Each thread returns only the result of executing the logic on its portion of the dataset. For example, summing a variable for a dataset that has been partitioned in nine pieces in CAS produces nine sums instead of one (CAS does the sum on the nine pieces independently, returning the sum for each partition). Ensuring code produces the correct output requires thinking through how to perform calculations like these on distributed data and what needs to be done to bring the results back together correctly.

Figure 3: Illustration of Multithreaded Data Step Processing



(Source: SAS 2019c)

CONTROLLING ORDERING AND PARTITIONING IN CAS

One of the main drivers of CAS's performance gains is distributed data. In some cases, taking care of this processing in advance can significantly reduce runtimes, as long as the right syntax is used to ensure CAS does not attempt to subsequently regroup and reorder the data.[3, 4]

[3] Benchmarking tests for the Synthetic Plan Microsimulation observed runtimes that were approximately twenty times faster for a simple DATA step when using data that had been partitioned and ordered in advance.
[4] The microsimulation was developed using SAS Viya 3.4.

Controlling partitioning and ordering was critical for the last step in the microsimulation, which involved applying the same set of operations to a large number of bootstrapped simulated populations. To calculate the key metrics, all the observations for a given population needed to be in the same partition and in a specific order. CAS can help users meet those requirements with BY-group processing. However, BY-group processing does not alter the in-memory dataset; in the default case, CAS groups and orders the data "on the fly" and performs the computation using a temporary partition that disappears after the DATA step finishes (SAS 2019b). If a subsequent operation requires the same BY group, the data will be re-grouped and re-ordered, incurring the performance cost again. To minimize runtime, the grouping and ordering needed to be maintained, not performed for each BY statement.

PROC CASUTIL's LOAD statement facilitates partitioning and ordering the data, but the key is making sure that CAS leverages those ordered partitions throughout the entire process. This requires syncing the variables used to partition, order, and process observations. The code snippet below shows how to control the grouping and ordering used by the BY statement during a CAS session. PROC CASUTIL's LOAD statement readies the data that gets used in the DATA step that follows. The variables in the GROUPBY= argument determine the partitioning. The ORDERBY= argument must contain that same set of variables, as well as any others needed to execute the logic in the DATA step. The variable list used in the DATA step BY statement must exactly match the list in the PROC CASUTIL ORDERBY= argument. Otherwise, it will trigger an on-the-fly partitioning of data as CAS treats the BY variables as a new group/order. Moreover, the PARTITION= and ORDERBY= data set options should be used in the DATA step output to preserve the partitioning and ordering.

```
libname mycaslib cas caslib="CASUSER";

* assuming we have a populations dataset in work;
proc casutil;
    load
        data=work.populations
        casout="populations_partitioned"
        groupby=(population_id)
        orderby=(population_id sort_order);
quit;

data mycaslib.simulation_output (partition=(population_id)
                                 orderby=(population_id sort_order);
    set mycaslib.populations_partitioned;
    by population_id sort_order;
    * additional steps follow that take advantage of the partitioning;
run;
```

INTEGRATING WITH OPEN-SOURCE SOFTWARE

The Synthetic Plan Microsimulation integrates open-source algorithms to augment the range of statistical and machine learning models used to forecast outcomes. This enhancement was possible because SAS Viya facilitates interfacing with open-source software, enabling algorithms created with SAS or languages like Python and R to be seamlessly integrated into the same code pipeline. In this case, the flexibility meant that the Synthetic Plan Microsimulation was able to take advantage of algorithms that were developed using Base SAS 9.4 procedures (e.g., PROC PHREG), SAS CAS procedures (e.g., PROC LOGSELECT), and machine learning packages from Python (e.g., keras.models and keras.layers).

Each model used by the microsimulation was developed independently with scoring procedures codified as a separate program; these programs are then brought together in a specific sequence to form the simulation pipeline. To execute this sequence from start to finish requires both SAS and Python. Rather than requiring a programmer to manually switch between SAS and Python as the pipeline executes, a program was written that could call each component sequentially without human intervention. This capability required understanding how to move between SAS and Python in SAS Viya, which leaned on a couple of key considerations highlighted below.

INTERFACING BETWEEN SAS AND PYTHON

For the pipeline to run successfully, the SAS and Python programs need to interface. The SAS program that governs the overall pipeline needs to (1) call the Python programs and (2) return information from the Python programs.

First, the SAS program (the parent program) needs to invoke a Python sessions in order to run a modeling component built in Python (the child programs). To do so, it employs SYSTASK command, which can be used by accessing the back-end console of the SAS Viya installation. The code example below shows the syntax for calling a Python program (child.py) using SYSTASK command. It applies the wait option to hold the execution of subsequent modeling components while the current Python component executes.

Second, the SAS program needs to be able to pull back information from Python about how the execution fared. The syntax below stores the status of the Python run in a macro variable (pystatus). This macro variable identifies whether the Python process ran correctly. If there is an issue in the Python execution, the syntax below makes sure the SAS log will include an error.

```
%macro call_python(parameter1=);
    * calls python via the command line passing the macro variable
      parameter1;
    systask command "python /path/to/child.py
        --parameter1 ""&parameter1."""
        wait taskname=pytask status=pystatus;

    * systask command stores the return value in a macro variable pystatus;
    * the program crashes if what python returns is not zero;
    %if &pystatus. > 0 %then %do;
        %put ERROR: Python encountered errors;
        abort;
    %end;
%mend;
```

MOVING DATA BETWEEN SAS AND PYTHON

In addition to needing the programs to interface, the pipeline also needs the data to move between SAS and Python. That includes two directions: from SAS to Python and from Python to SAS.

From SAS to Python: When the pipeline reaches a model written in Python, data that is currently loaded into memory needs to be accessed by Python. In Python, the SAS Scripting Wrapper for Analytics Transfer (SWAT) package can be used to directly query data in CAS (SAS Institute 2017). Alternatively, data that has been loaded into CAS (.sashdat) can be converted to a SAS dataset (.sas7bdat) so that it can be read in using Python’s pandas package (pandas.read_sas[5]). Converting a file from .sashdat to .sas7bdat can be achieved in a DATA step that reads from CAS and writes to SPRE:

```
libname mycaslib cas caslib="CASUSER";
```

[5] https://pandas.pydata.org/docs/reference/api/pandas.read_sas.html

```
libname pyinput "/path/to/python/input/dataset";

data pyinput.limited_vars_data (keep=&keep_vars.);
    set mycaslib.pipeline_data;
run;
```

From Python to SAS: To continue running the pipeline after the Python model finishes, the output from Python needs to be loaded back into CAS. To make this transition, the pipeline's Python programs end with a code snippet that takes an output comma separated values file (CSV) and uploads it to CAS to continue processing in SAS. This transfer can be accomplished in a couple ways, but the data will follow a progression from a pandas DataFrame[6] to CSV to CAS (Smith 2017). This code snippet shows one way to write the output of the Python script back to CAS from Python:

```
# creates connection to cas using swat package
# this assumes host, port, and authinfo are all defined
conn = swat.CAS(host, port, authinfo, protocol='cas')

python_output = '/path/to/python/output/predictions.csv'
cas_dataset_name = 'predictions'

# action set to take care of loading the table loading into cas
conn.loadactionset(actionset='table')

# uploads the new python output to cas as cas_dataset_name
tbl = conn.upload_file(python_output,
    casout={'name':cas_dataset_name}, promote='True')

conn.close()
```

The file is loaded to the user's CAS library, which can then be accessed in SAS to continue subsequent steps. The ability to move data directly between Python sessions and CAS improves the efficiency of interacting between open source languages and SAS.

USING INTERACTIVE VISUALIZATIONS

As described above, the Synthetic Plan Microsimulation is intended to enable business users to make better decisions regarding the RADV program. To assist with this goal, business users explore the results of the microsimulation through an interactive data visualization portal. This SAS-based web portal allows the user to compare scenarios – where slight modifications in assumptions result in different outputs. As described above, each scenario is a hypothetical 'what if' analysis, allowing the business user to select and compare strategies and see how the outcome metrics could change under these potential adjustments. For example, if a user wants to estimate the proportion of Medicare Advantage beneficiaries with diabetes, the Synthetic Plan Microsimulation can generate scenarios under multiple possible prevalence rates (e.g., rates of 25%, 50% and 75% MA beneficiaries with diabetes), and estimate confidence intervals for key metrics using the bootstrapping procedure. Business users can review these results and consider the implications on behavioral changes, disease prevalence, or beneficiary behaviors, as well as begin to assess the impacts of potential programmatic changes.

In order to provide a user with an interactive experience, the Synthetic Plan was designed to enable the user to explore results through interactive visualizations built in SAS Visual Analytics. The simulation pipeline outputs a series of SAS datasets (i.e., .sas7bdat files) which are stored for import into SAS Visual Analytics and enables a user to interactively

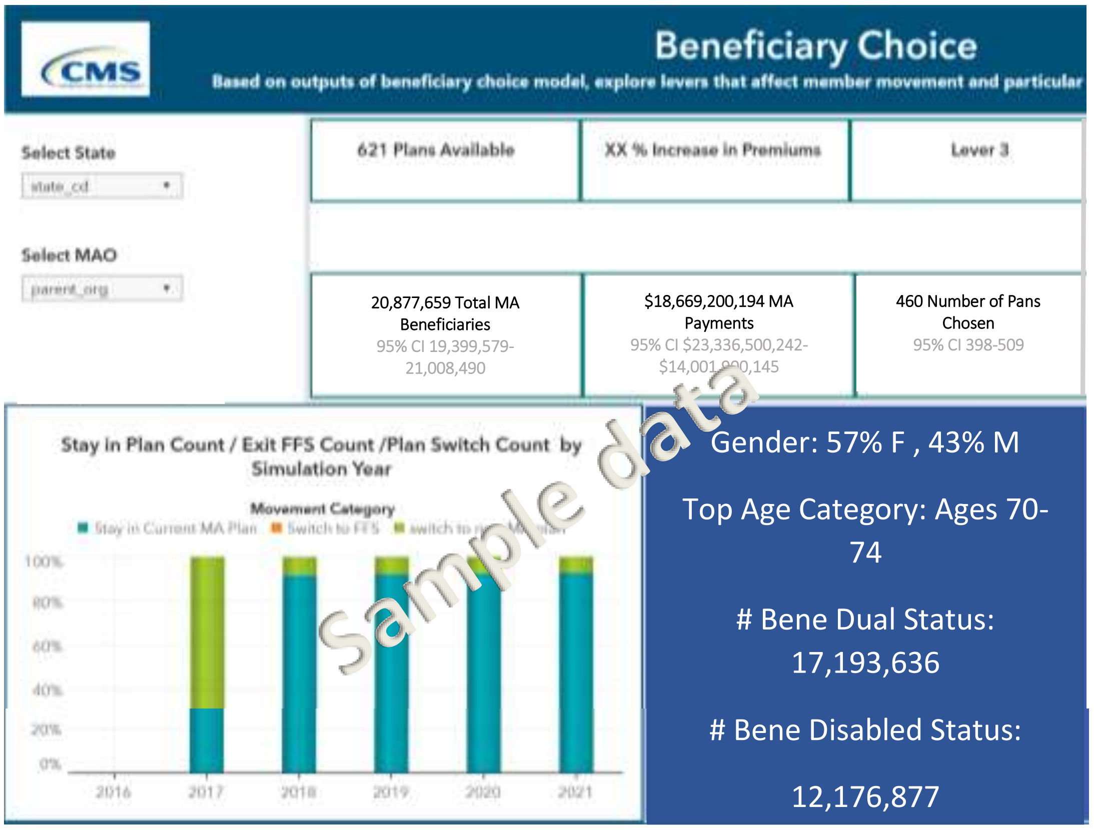
[6] A pandas DataFrame is analogous to a SAS dataset.

explore results. Business users have the ability to interact with the results, slicing data into subgroups to identify areas for additional analysis or research. SAS Visual Analytics has the ability to integrate SAS datasets as CAS datasets, which can be stored and used as in-memory datasets for additional scenarios more efficiently. Having the data easily integrated into the visualization output allows for a rapid pipeline, enabling rapid evaluation of multiple scenarios and visualization of the results.

A data flow process brings data from relevant modular components to the visualization layer using a scenario naming convention, enabling administrators of the Synthetic Plan to track scenarios run through the system. Business users review these scenarios, seeking patterns in the data by applying filtering criteria to the data at various levels of granularity (e.g., individual person, group, geographic area). For example, Figure 4 displays a view illustrating scenarios about how beneficiaries make choices about choosing Traditional Medicare versus a Medicare Advantage plans available to them.

In this view, the user is presented with patterns from a model that estimates the movement of Medicare beneficiaries both within Medicare Advantage as well as from Medicare Advantage to Traditional Medicare Fee-for-Service (FFS). There are two filters on the upper left-hand corner, allowing the user to select the state of interest and Parent organization. The top three boxes illustrate scenarios that the business user could modify to estimate the impact on beneficiary choice outcomes. For example, predicting how the increase in availability of Medicare Advantage plan choices impacts each beneficiary's predicted likelihood of staying in their current plan, switching to another plan or switching to Fee-for-Service. The three boxes underneath characterize the volume of beneficiaries, payments and plans included in the scenario. This information assists the user by adding context to the results. For example, if all beneficiaries were predicted to switch from Medicare Advantage to FFS that might seem anomalous, unless the visualization indicated only for a small subset of people were selected. The stacked bar chart identifies trends over the five-year forecast, to demonstrate how the population of Medicare Advantage beneficiaries may be changing. Users can compare to values to those from external sources for additional validation. The bottom blue text box illustrates the characteristics of a particular sub-group (e.g., characteristics of the beneficiaries most likely to switch to FFS). Note that data displayed in Figure 4 are test data for illustrative purposes only. Real values are masked in this paper.

Figure 4. Beneficiary Choice view: a scenario-focused view



621 Plans Available

XX % Increase in Premiums

Lever 3

20,877,659 Total MA Beneficiaries
95% CI 19,399,579-21,008,490

$18,669,200,194 MA Payments
95% CI $23,336,500,242-$14,001,900,145

460 Number of Pans Chosen
95% CI 398-509

Gender: 57% F , 43% M

Top Age Category: Ages 70-74

# Bene Dual Status: 17,193,636

# Bene Disabled Status: 12,176,877

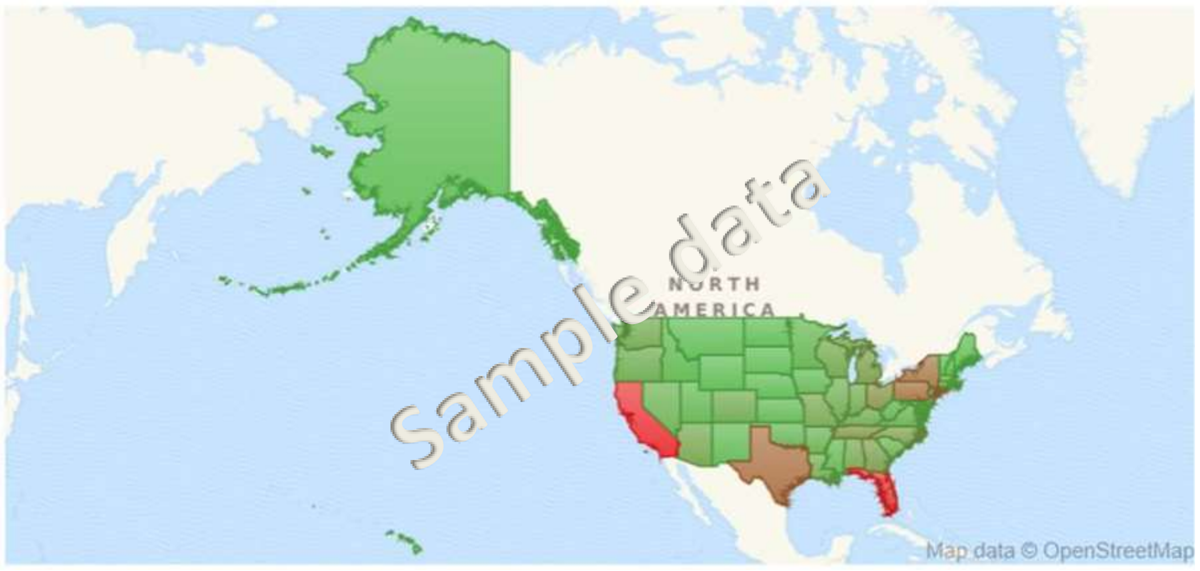
Multiple views were designed and built, enabling users to explore different facets of the microsimulation outputs, including healthcare purchasing decisions, potential future health of the population, and risk of payment error. Since scenarios help business users to derive insights, the dashboard views were designed to include content in a clear and easily digestible manner. Some designs required workarounds for the standard widgets to enhance the aesthetics and utility of the outputs. This paper will discuss two of these workarounds:

1. United States geographic mapping
2. Text box displays using data categories

U.S. GEOGRAPHIC MAPPING

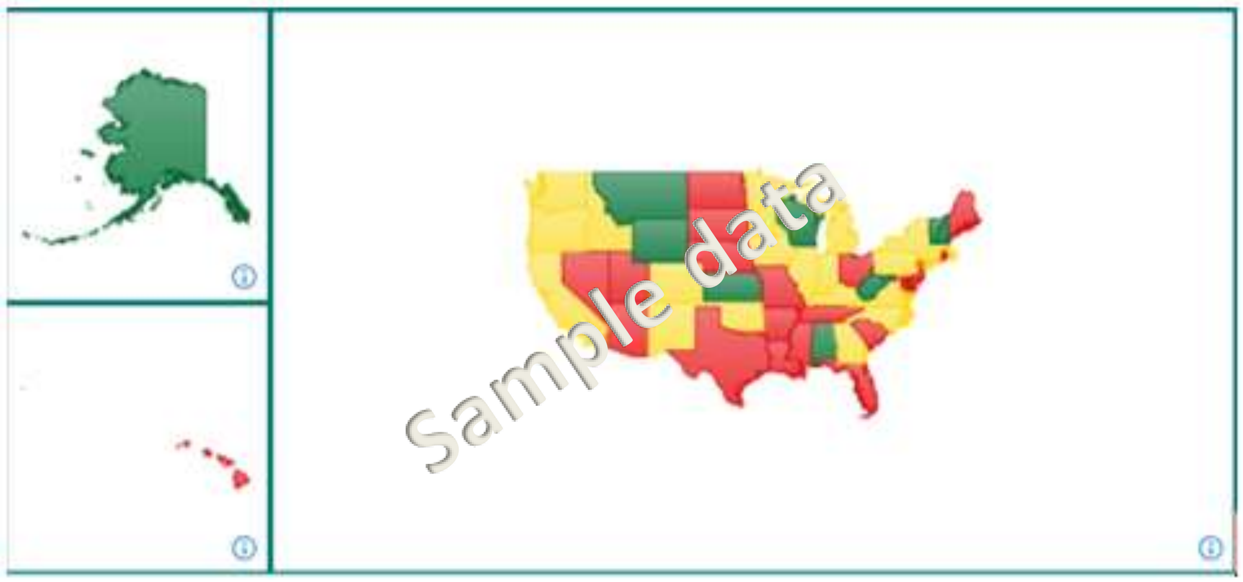
Many scenarios required the consideration of geographic trends and variation. Healthcare decisions and behaviors can be driven by sub-populations within a given geography making geographic widgets necessary for this work. When incorporating the U.S. state-level map, the widget includes Alaska, Hawaii and the contiguous states. Given Alaska and Hawaii's size relative to the rest of the country (Alaska much larger, and Hawaii much smaller, respectively), a state-based widget was not aesthetically pleasing as Alaska took up most of the space in the widget, making navigation and interpretation difficult (Figure 5). In this format, the user must use the hand tool to navigate across the panel and zoom in on the contiguous United States (e.g., zoom in, zoom out, center the area of interest). It is also difficult in this default view for the user to hover over particular states for information.

Figure 5. U.S. Geographic Map Single Panel



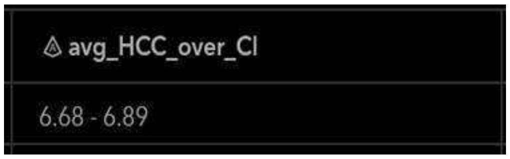
To mitigate this navigational issue for the user, three different side-by-side widgets were created to illustrate the look of a complete U.S. map but in a visually pleasing way (Figure 6). In this format, the user can easily see the trends of the contiguous U.S. without having to use the hand tool to navigate within the widget.

Figure 6. U.S. Geographic Map Multi-Panel



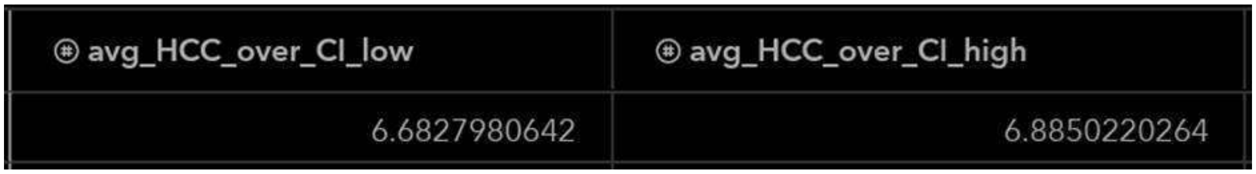
TEXT BOX DISPLAYS USING DATA CATEGORIES

For some displays, additional descriptions assist the user in understanding the context around the numbers presented. The descriptions needed to be inserted as text box widgets into the visualization. Displaying confidence intervals which is in the format of two numbers separated by a dash '-' is important in communicating that 95% of the time the estimates should fall within that given range of values. As a text box widget was included to display the confidence interval, the value was originally represented in a single column with character type. For example:



avg_HCC_over_CI
6.68 - 6.89

Only character values can be displayed in a text box that can only accept dynamic values from numeric fields. The visualizations have different metrics with different associated confidence intervals, therefore the text box value for a confidence interval needed to be updated based on the metric being displayed. Another challenge was that different metrics of interest may have different numerical formats. For example, some metrics may represent a count of health conditions while another metric may represent monetary values (requiring a dollar sign format before the value and a two-digit value following a decimal). To mitigate this issue, the lower and upper bounds of the confidence interval were separated into two different data columns, both numeric data types:



avg_HCC_over_CI_low	avg_HCC_over_CI_high
6.6827980642	6.8850220264

After separating the lower and upper bounds into separate columns in the underlying data structure, these were imported into the text box widget, with a '-'separating the numerical inputs. Since the variable formats, properties and labels can be used to specify whether the value is a numeric integer compared to a monetary value, the dynamic definition of 'number "-" number' can be easily replicated for all numeric types of data.

SAS Visual Analytics can be a tool that offers an interactive platform for business users. When combined with SAS Viya tools (e.g., SAS Studio), Visual Analytics is a helpful companion to quickly integrate outputs from models into interactive views. However, SAS Visual Analytics designers may face challenges in designing a view with multiple filters and widgets—especially in sizing the widgets together. For example, as described above, when including a geographic map widget, separate panels can be included to ensure the sizing relative to other geographic elements. Text box information must be in the form of a measure value so any values of interest must be used as a measure first. The workarounds provided mitigate some of the aesthetic challenges faced during the development of the Synthetic Plan platform.

CONCLUSION

This paper describes how the Synthetic Plan Microsimulation was implemented in SAS Viya including components for a Big Data Framework, Integration with Open Source Tools, and Interactive Visualizations. This combination of open source and SAS tools such as CAS and Visual Analytics make it possible for analysts to build complex microsimulations systems and help business users draw conclusions from data. We hope the information in this paper can assist SAS users in producing useful tools for better decision making.

REFERENCES

Baker J., Shnaiden, T., and Thompson E. 2017. "Demographic Microsimulation in Wellness and Health Promotion: Machine Learning Meets Applied Demography." *Annual Meeting of Population Association of America*.

Barton, Russell R. and Schruben L.W. 1993. "Uniform and bootstrap resampling of empirical distributions." *Proceedings of 1993 Winter Simulation Conference-(WSC'93)*.

Blumberg, L. J. 2003. "*The Health Insurance Reform Simulation Model (HIRSM): Methodological Detail and Prototypical Simulation Results*."

Bultman, D. & Secosky, J. 2018. "Parallel Programming with the DATA Step: Next Steps." *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc. Available at: https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2184-2018.pdf.

Efron, B. 1982. "The jackknife, the bootstrap, and other resampling plans." Vol. 38.

Efron, B. 1987. "Better bootstrap confidence intervals." *Journal of the American statistical Association,* 82.397: 171-185.

Glied, S., and Tilipman, N. 2010. "Simulation modeling of health care policy." *Annual Review of Public Health,* 31: 439-455.

Lewin Group. 2009. Health Benefits Simulation Model (HBSM): Methodology and Assumptions. Available at: http://www.lewin.com/content/dam/Lewin/Resources/Site_Sections/HBSMSummary.pdf

Lomax, M and Smith, AP. 2017. "An introduction to microsimulation for demography." *Australian Population Studies*. 1(1): 73-85.

Marois, G and Belanger, A. 2014. "Microsimulation Model Projecting Small Area Populations Using Contextual Variables: An Application to the Montreal Metropolitan Area, 2006-2031." *International Journal of Microsimulation*. 7(1): 158-193.

Mills, A. 2013. "Simulating health behavior: A guide to solving complex health system problems with agent-based simulation modeling". Available at: https://www.soa.org/resources/research-reports/2013/simulating-health-behavior-a-guide-to-solving/

Rutter C., Zaslavsky A., & Feuer E. 2011. Dynamic Microsimulation Models for Health Outcomes: A Review. *Medical Decision Making*, 31:10-18.

Walsh, I. "Pros and Cons of X command vs. SYSTASK command." 2009. Available at https://www.lexjansen.com/wuss/2009/cod/COD-Walsh.pdf

Pendergrass, J. 2017. "The Architecture of the SAS Cloud Analytic Services in SAS Viya^{TM}." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available at: https://support.sas.com/resources/papers/proceedings17/SAS0309-2017.pdf.

SAS Institute Inc. "Procedures That Use CAS Actions." 2019. Available at: https://go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.3&docsetId=proc&docsetTarget=p0nnkdmqmz48w8n1kqofzc7mcla4.htm&locale=en.

SAS Institute Inc. "BY-Group Processing in CAS." 2019. Available at: https://go.documentation.sas.com/?docsetId=casdspgm&docsetTarget=n0jhhmgs0eiifen1djfd4go9mn8a.htm&docsetVersion=3.5&locale=en.

SAS Institute Inc. "Processing Modes." 2019. Available at: https://go.documentation.sas.com/?docsetId=casdspgm&docsetTarget=n183xun5dd4dy5n1x453fw5reu8o.htm&docsetVersion=3.5&locale=en#n1ais5ys5mivqpn1a7kw7swjxao2.

SAS Institute Inc. "Running the DATA Step in CAS." 2019. Available at: https://go.documentation.sas.com/?docsetId=casdspgm&docsetTarget=n046ex9crhf1yqn16anyzibwbw0v.htm&docsetVersion=3.5&locale=en.

Secosky, J. 2017. "DATA Step in SAS Viya^{TM}: Essential New Features." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available at: http://support.sas.com/resources/papers/proceedings17/SAS0118-2017.pdf.

Shannon, D. 2018. "Come On Baby, Light my SAS Viya: Programming for CAS." *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc. Available at: https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2622-2018.pdf.

Sober, S. 2018. “My Experiences in Adopting SAS Cloud Analytic Services into Base SAS Processes.” *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc. Available at: https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1710-2018.pdf.

Weiss, P. “Getting Your SAS 9 Code to Run Multi-Threaded in SAS Viya 3.3.” 2018. Available at: https://blogs.sas.com/content/sgf/files/2018/01/TechnicalWhitePaper_SASViya_Jan2018_FINAL.pdf.

ACKNOWLEDGMENTS

This paper was a collaboration, including the work of many individual people. The authors would like to thank all the people who contributed to this paper and those who contributed to the Synthetic Plan Microsimulation. In particular, we would like to thank Antonio Soriano, Gulnur Freeman, Chris Parker, Larry Johnson, Evan Boyarsky, Sally D’Amato, Ritesh Verma, Julie Leonard, Satya Settipali, Sunny Verma, Jonathan Owens, Craig Forman, Evan Sands, Ashley Hall, Andre Lam, Michael Bowerman, Michael Liou, Nick Toure, Antonio Zavaleta, Ben Weiner, Zachary Rom, Rima Gooden, Bill Roberts, Crispen Masunda, Natasa Kordic, Iñaki Arbeloa, Lateefah Hughes, Sadie Bennett, Yancy Dennis, Matt Honkanen, Rob Williams, Jessica Leaman, Jack Baker, and Hans Dutt. We would also like to thank our colleagues from SAS who helped make this possible including Manuel Figallo and others who have contributed their knowledge and experience. In addition to those mentioned, many others contributed as well, and we thank you.

CONTACT INFORMATION

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

Jonathan Smith
Jonathan.smith@cms.hhs.gov