SAS3368-2019

Deploying Models Using SAS® and Open Source

Jared Dean, SAS Institute Inc.

ABSTRACT

In the excitement and hype around machine learning (ML) and artificial intelligence (AI) most of the time is spend in the model building. Much less energy is expended on how to take the insights from models and deploy them efficiently to create value and improve business outcomes.
This paper will show a complete example using DevOps principals for building models and deploying them using SAS® in conjunction with opens source projects including Docker, Flask, Jenkins, Jupyter, and Python. The reference application is a recommendation engine on a web property with a global user base. This use case forces us to confront security, latency, scalability, repeatability. The paper will outline the final solution but also include some of the problems encountered along the way that informed the final solution.

INTRODUCTION

SAS Communities is a peer-to-peer community for SAS users to ask questions and find answers from each other and from experts at SAS. To improve the experience of users on the site we wanted to create personalized recommendations. To help visitors find articles of interest among the tens of thousands of active articles on the site. In Figure 1 Screen Shot of Recommendations you can see the finished product. I am the first to acknowledge that this does not make the most exciting demo but there is coordination between different personas and technologies to make the recommendation experience transparent to the end user.

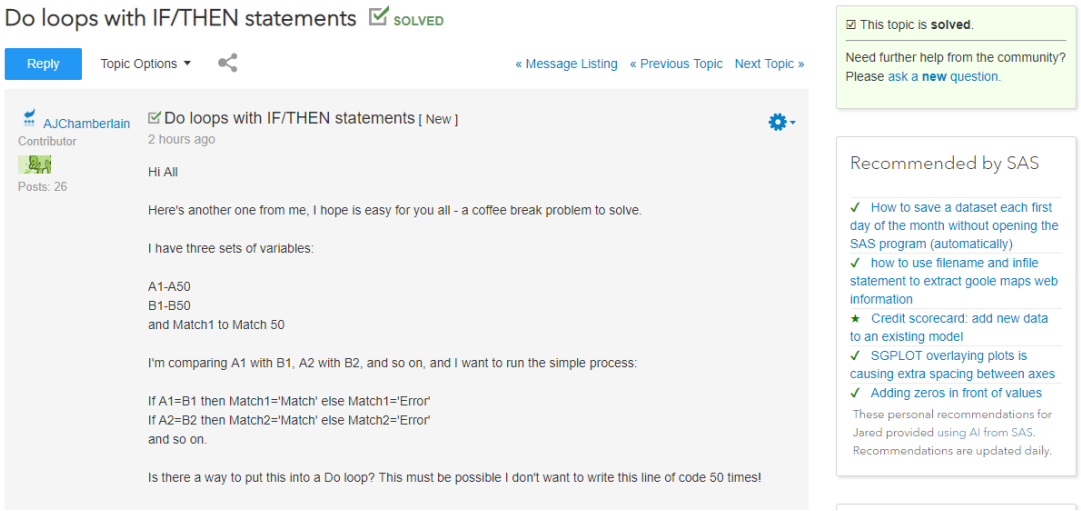

Figure 1 Screen Shot of Recommendations

My goals in writing this paper are: first is to demonstrate how SAS is using its technology to solve its own business challenges of improving the user experience on our SAS Communities web property. Second, to provide a template for how you can implement these ideas in your organization either a web property or some other service that would benefit from recommendations. Next, to highlight how SAS works with open source tools. Finally, how SAS can be used in a cloud ready environment without long installation and configuration times.

From the software point of view, this project uses several SAS products along with open source tools. SAS and open source tools work in concert with each other to help users be more productive and impactful to their organizations.

From the human capital perspective, this project involved several personas. A data scientist to prepare the data and build the models, a systems engineer to deploy and monitor the model on a daily basis, a security engineer to help ensure compliance and safety of our applications, a full stack developer to integrate the API call into the website.

ARCHITECTURE AND TOOLS

This project uses SAS DATA step for the data prep and the FACTMAC procedure, as part of SAS Visual Data Mining and Machine Learning (VDDML) for the recommendation model.

Table 1. Utilized Open Source Tools is a list of the Open source tools used across the project. If you're a SAS user, then many or even all the tools in the table might be unfamiliar. If your first impression is that the table is rather long, I agree. Integration with open source tools will bring you into contact with many projects all with different levels of maturity and style and function. Don't be intimidated, keep in mind this project spans several personas and many of these tools are standard and well known across IT / systems professionals. It would be very unusual for a single person to use all these tools.

Table 1. Utilized Open Source Tools

Open Source Tool	Description
Jupyter	Web Based IDE; supports many kernels including Python and SAS.
SASPy	Open Source project that allows python programmers to use the SAS computing engine (9.4m0 or newer).
Python-SWAT	Open Source project that allows python programmers to use the SAS Viya computing engine.
Docker	Container technology that allows quick consistent deployment of environments.
Kubernetes	Open source project to manage and scale docker containers.
OpenStack	Private cloud infrastructure.
Terraform	Orchestration tool for quickly deploying cloud assets.
Git	Source management tool.
Python	Open Source programming language; object oriented; created in 1985.

Analytics professions have been building models for years and years. It is the fun and exciting part of the job for most nerds like me. The part that is often manual time consuming and costly to the organization. This paper demonstrates how SAS can be used, along with open source tools, to create a modeling and deployment process that follows the Continuous Integration/ Continuous Deployment (CI/CD) methodology

Figure 2. SAS Community Recommendation Process illustrates the process that I have used to create personalized recommendations for the SAS Community users based on their past browsing history. The Build box from Figure 2 is the domain of the modeler and you can read more about that role in this project in the Modeling section.

While I had the main responsibility for this project, no successful project is done alone. Partnering with colleagues within your organization and leveraging their strengths and expertise will reduce implementation time and cost to the organization.

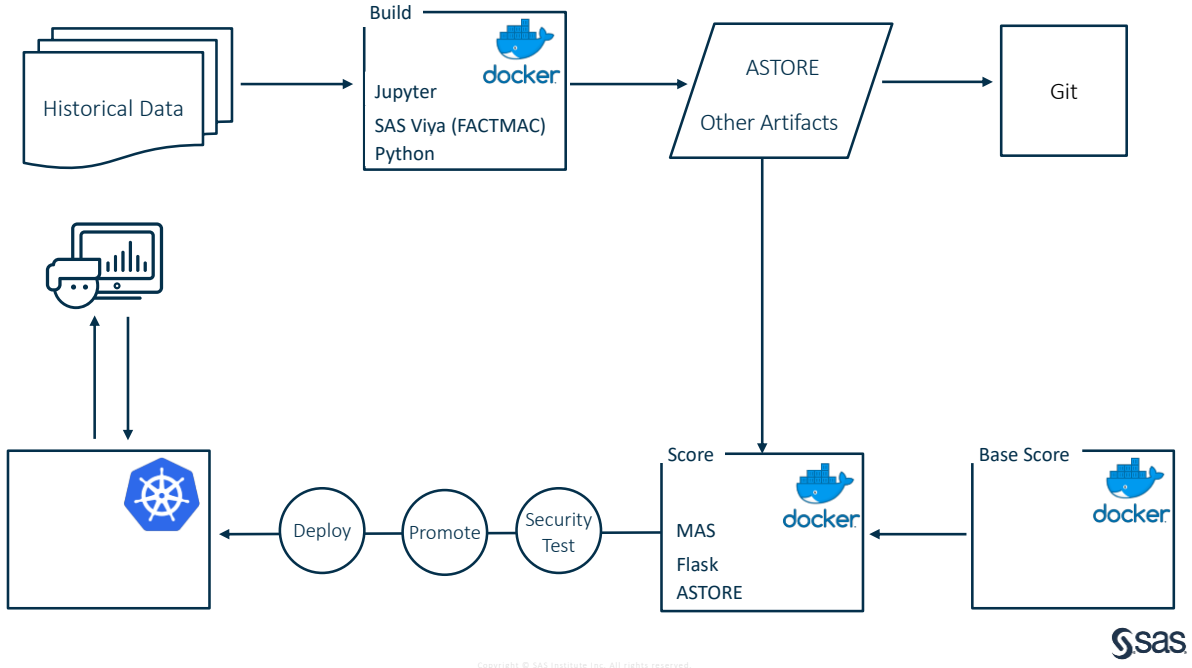

Figure 2. SAS Community Recommendation Process

The

MODELING

This is primary domain for the data scientist (or appropriate vogue business title). They have three main responsibilities: prepare the data, build a great model, generate the artifacts for packaging and deployment.

PREPARING THE DATA

There are several required steps in preparing the data for modeling. The size of the and frequency of this problem posed several challenges in this area. The first challenge is the data size. SAS Communities is a very popular site and the transactional historical data is large. Using data from late 2015 to present is about 40GB of data. So the first task is to remove the variables and rows that do not relate to our modeling problem.

The next challenge is creating ratings. In most cases users provide recommendations on the products or services they consume. Netflix ratings on movies is one example in this case we do not have recommendations, but machine learning techniques are not very effective if there is no discrimination between articles. So, we must augment the data using a process called implicit feedback in which we will take all the articles a user has seen and then randomly sample from a set of articles they have not seen to get two equal sized partitions. The viewed articles receive a rating of 1 and the non-viewed articles receive a rating of 0. The coding challenge is to extract the articles viewed by a particular user and then sample without replacement from non-viewed articles in an efficient manner. The code must be as efficient as possible because there are over 130,000 registered members of the SAS Community and this process is run every day to provide the most up to date recommendations. To illustrate the speed requirement if the process took just one second per user that would be more than 36 hours if done serially so parallelization and efficiency are paramount.

In using SAS 9.4m6 I was able to reduce the data prep to 23 minutes through macros and hash tables. I then moved the DATA step code to CAS DATA step and the time was reduced to just over 3 minutes on the same hardware. CAS DATA step has the added benefit in this case that when the code moved to a cloud deployment it maximizes the computing for whatever size system it runs on without additional code modifications on my part.

Here is my code for implicit feedback per user:

```
data mycas.implict_feedback;
    array found[&nconv.] _temporary;
    array compressed[&nconv.] _temporary;
    set mycas.conversations;
    by user_uid;
    retain count;
    if first.user_uid then do;
        count=0;
        do i = 1 to dim(found);
          found[i]=0;
       end;
    end;
    found[conversation_ord] = 1;
    rating=1;
    count+1;
      output;
  The code above initializes two temporary arrays to store the conversations
  a specific user has viewed from the universe of all conversations. The size
  of the array &nconv is the number of unique conversations at the time of
  modeling. Every article is output with a rating of 1.

  The code below runs after we have a complete list of the conversations
  viewed by a user. It randomly selects a set of conversations that a user
  has not viewed and sets their rating to 0. The resulting dataset,
  mycas.implicit_feedback, has exactly twice as many observations as the
  starting dataset, mycas.conversations.

 if last.user_uid then do;
       unseen = 0;
       seen = 0;
       /* make the compressed array of unseen conversations */
       do i = 1 to dim(found);
           if found[i]=0 then do;
               unseen+1;
```

```
                compressed[unseen] = i;
            end;
            else do;
                conversation_ord = i;
                seen+1;
            end;
        end;
        /* shuffle them */
        retsize = min(count,max(seen,unseen));
        if seen > unseen then put user_uid= seen= unseen=;
        do i = 1 to retsize;
            j = floor(rand('uniform')*unseen)+1;
            temp = compressed[i];
            compressed[i] = compressed[j];
            compressed[j] = temp;
        end;
        do i = 1 to retsize;
            conversation_ord = compressed[i];
            rating = 0;
            output;
        end;
    end;
run;
```

BUILDING THE BEST MODEL

With the data prepared we can move to modeling. Part of the business requirements for the recommendations were to create recommendations that favor articles with accepted solutions and favor newer articles. The logic for this is that accepted solutions will be more useful to users and the articles you've viewed recently are more applicable to your current interests than those from several years ago. The FACTMAC procedure doesn't have a weight statement yet so I created duplicate rows to nudge PROC FACTMAC to follow these requirements.

Here is the SAS code to weight data by both recency and having an accepted solution:

```
%let wf=3;
data mycas.weighted_factmac / single=no;
    set mycas.implict_feedback;
    if _n_=1 then do;
        declare hash conv (dataset:'mycas.conversation_uid_index');
        conv.DefineKey('conversation_ord');
        conv.DefineData('start_id', 'end_id');
        conv.DefineDone();

        declare hash row_lookup (dataset: "mycas.conversations");
        row_lookup.DefineKey('id');
        row_lookup.DefineData('event_time_ms', 'isSolvedTopic');
        row_lookup.DefineDone();
    end;
    if rating=0 then do;
        conv.find();
        id = (start_id + floor((1+end_id-start_id)*rand("uniform")));
        row_lookup.find();
    end;
    decay = (datepart(event_time_ms)-&mindate.)/(&maxdate. - &mindate.);
```

```
    rep = int((1-decay)*10);
    daysback = intck("DAYS", datepart(event_time_ms), today(), "C");
    do i = 1 to rep*&wf.;
        output;
    end;
  run;
```

Using HASH objects I can quickly look up the information needed to properly weight the observations.

One important feature of FACTMAC is the autotuning. The AUTOTUNE feature uses optimization to search for the best hyperparameters saving me time hunting for the ideal combination of settings (that could change over time). The running of this model takes hundreds of CPU hours to complete the search for the best hyperparameters but it saves much more than that in human capital costs. For more information on AUTOTUNE see the suggested reading section.

In the latest iteration, FACTMAC modeling using the AUTOTUNE statement costs about 300 CPU hours of time and because the problem is only changing slightly each day (one new day of data among more than 1000 days) I reduced the daily run time by using some of the options available in AUTOTUNE. Here is my code for the FACTMAC procedure:

```
proc factmac data=mycas.weighted_factmac  outmodel=mycas.factors_out;
   autotune maxtime=3600 objective=MSE
        TUNINGPARAMETERS=(nfactors(init=20) maxiter(init=200)
learnstep(init=0.001));
   input user_uid conversation_uid /level=nominal;
   target rating /level=interval;
   savestate rstore=mycas.sascomm_rstore;
run;
```

I use the MAXTIME option to limit the search to one hour and I use the TUNINPARAMETERS option to start with best configuration from my last complete run (which I run periodically). This strategy gives me the best known hyperparameters within a time budget and an opportunity to find even better hyperparameters with the extra time.

CREATING ARTIFACTS

With the data prep and modeling complete I can now create all the needed artifacts to quickly and efficiently deploy a scoring model. The main artifact I need is an ASTORE (see suggested reading for more information) which is a portable compressed binary object that contains the scoring logic to predict how much a specific user would enjoy a specific conversation on the SAS Community. In addition to the ASTORE, Table 2. Modeling Artifacts details the items created.

Table 2. Modeling Artifacts

Artifact	Purpose
Data set of active articles	To speed the scoring with a preloaded list
Json file of most popular articles	This is the fallback recommendation when nothing better can be presented

Artifact	Purpose
Articles viewed in the last N days	We don't want to continue to recommend the same things over and over so if you've looked at an article we won't recommend it for a while.
Data file with keys and timestamps	This file is used to ensure files are transferred correctly and for reporting information at score time.
FACTMAC ASTORE	Compressed binary container the scoring logic for each user and item.

The artifacts from Table 2. Modeling Artifacts are produced daily because we get updates to the data daily. These artifacts are packaged into a Docker container they could also be placed under source management in Git.

ORCHESTRATION

The orchestration of this process has evolved over time. It is my recommendation that you not try to automate and script everything in the beginning but instead continuously improve the process over time in progressive steps. This is one of the main philosophies of CI/CD that I think is well aligned with how projects develop and mature over time.

This project started out with just a Jupyter notebook to create the artifacts. Figure 3 Improvement of Process to reach Cloud Ready show the evolution toward fully cloud deployed and autonomous. Here are the written details:

After the initial model was created, I encapsulated the notebook in a Docker container to ensure software stability from run to run. After that, I created a bash script to run the steps via the Unix utility cron instead of running the steps manually each day. I was only ready for cron once the process was repeatable and stable. I expected each step to work, I was not hoping it would work. With the bash script I could automate the process to run daily even when I was out of the office. Over time, I needed better monitoring. I moved my bash script into Jenkins, an open source orchestration tool use by SAS R&D. After a few weeks it was decided the project should be managed by IT not R&D so I needed to migrate my script from Jenkins to Bamboo (a comparable product from Atlassian) which was the orchestration tool of choice for the IT team. When I moved the project to Bamboo, we discovered that the bamboo agents (servers available for tasks) were not sufficiently large to run my project. So, I evolved the project one last time to initiate the hardware resources in OpenStack (an open source cloud operating system) using Terraform.

I share these details with you to demonstrate that the result of a fully scalable cloud deployment of SAS to prepare, model, and score was not the initial release instead it was an evolution to meet the project objectives. The project can now access large amounts of computing power for a short period of time and upon completion release the computing power to others.

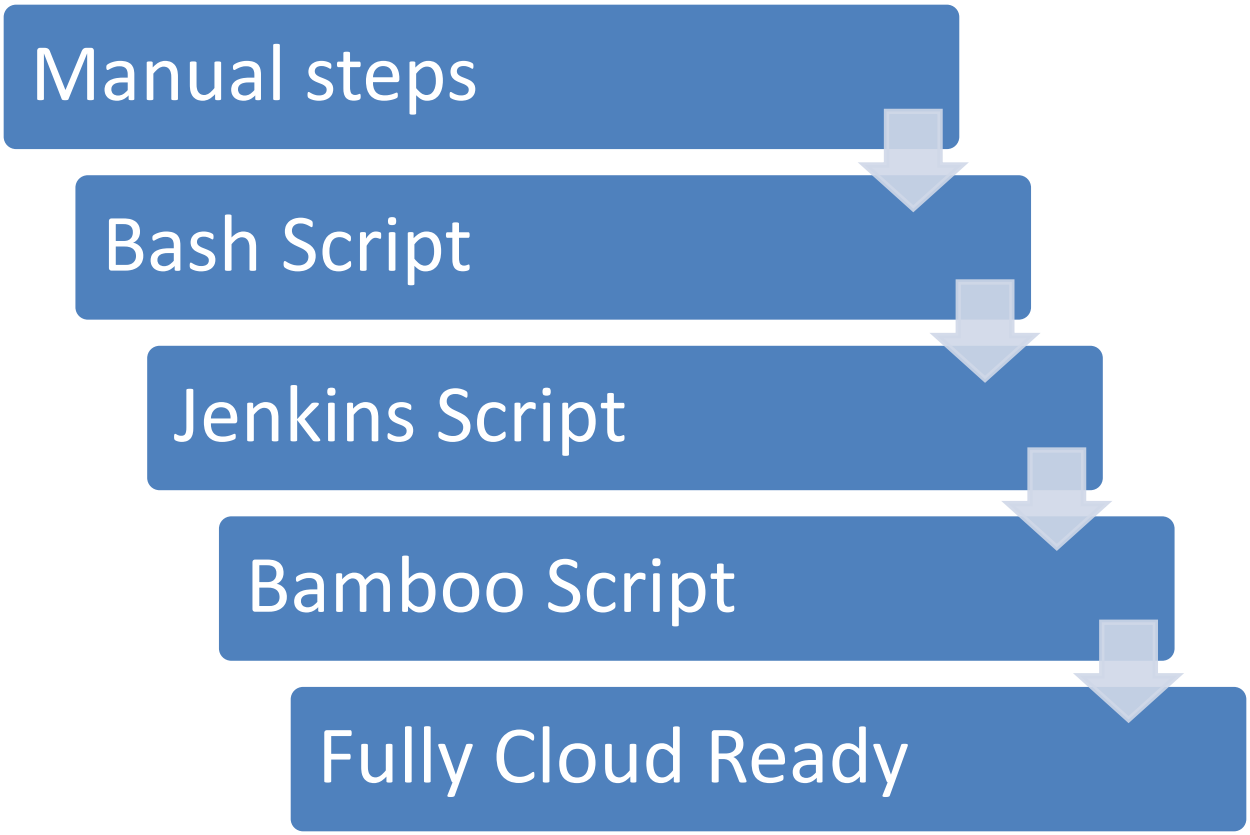

Figure 3 Improvement of Process to reach Cloud Ready

DEPLOYMENT

The deployment aspect of this project was the newest to me. Because of the need to run daily, a robust automated deployment process without an outage was required. We wanted users to have personalized recommendations with less than a two second delay. Working with the IT staff, the best practice is to build a scoring container in docker and then manage that container via Kubernetes. Docker was also used to build the model artifacts because it guaranteed an immutable environment and portability to different hosts.

The scoring docker container is different from the modeling building container in a few significant ways. The first difference is that the scoring container is purpose built to score just one model, in this case, personalized recommendations for SAS Community users. The model building container has a complete version of SAS along with visualization tools. This makes the sizes very different. The scoring container is around 800MB and the model building container is around 13 GB. The smaller container makes it easier to deploy and scale. This smaller container also adds a measure of security because the scoring container is exposed to the internet (with several security measures) but has only the essential components needed to score incoming requests not a general-purpose workflow.

Inside the scoring container we have a Flask application running under Gunicorn. These are both open source projects written in Python. The Flask application receives the incoming API request and sends it to the ASTORE through the micro analytics service (MAS). MAS is a capability through SAS Model Manager or SAS Decision Manager. By using MAS, you can achieve very low latency scoring. When a user hits the SAS Community webpage while logged in, a request is sent to Kubernetes cluster which is running the scoring docker

containers. It is then routed to a specific container and received by the flask application. The flask application sends the userid and the number of items requested *N* (the default is 5; max is 25). To return the best *N* article recommendations, all active articles must be scored (around 85,000 minus any articles that the user has viewed in the cooling off period). After all the appropriate articles have been scored, it sorts the list by the users predicted rating and takes the top *N*. This typically happens in about 200ms which means it is performing about 400,000 transactions per second per core after initial startup. The actual flow of information is relatively simple. A request is made from a user by viewing a SAS Community webpage and the request is validated by the firewall and routed to a scoring container running in Kubernetes (known as a pod) the scoring container processes the request and then returns the top *n* items back as a json string to the web page all in about 200ms. See Figure 4. Scoring Infrastructure.

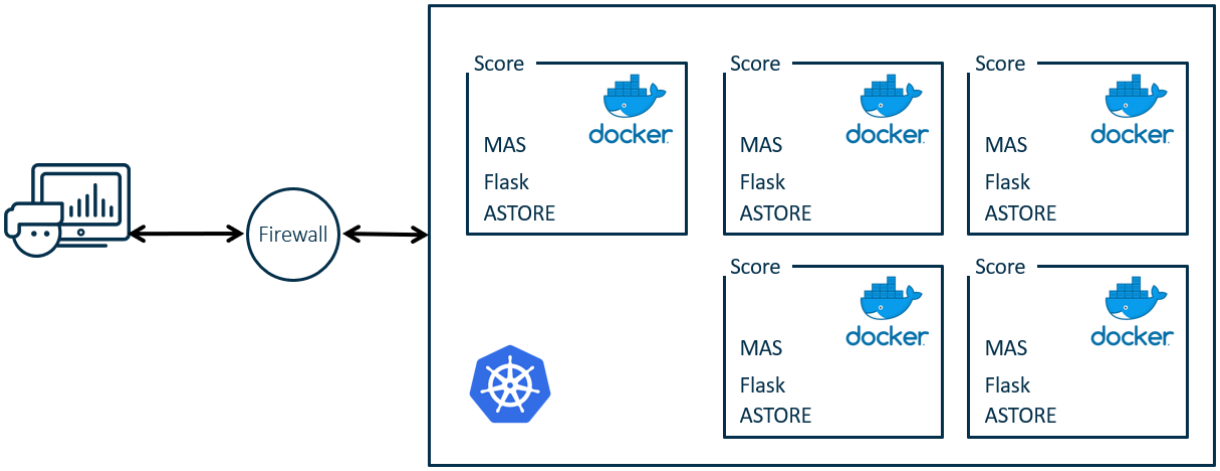

Figure 4. Scoring Infrastructure

CONCLUSION

This paper has detailed how SAS created personalized recommendations for the SAS Communities web property using SAS technology along with open source tools. This project uses a CI/CD framework that allows for automated daily updates to recommendations so that users are always getting the most current recommendations possible.

The recommendations are using cloud infrastructure to eliminate persistent hardware requirements for building the daily model and it can be scaled to meet the desired time constraints. The runtime scoring is using standard IT tools, Docker and Kubernetes, to fit seamlessly into workflows.

The SAS system integrates with open source tools to provide leading edge analytics in a cloud ready environment.

REFERENCES

Laster, Brent "What is CI/CD?" opensource.com 06 Aug 2018 Accessed February 22, 2019 https://opensource.com/article/18/8/what-cicd

Dean, Jared. "How to Build a Recommendation Engine Using SAS® Viya®." Proceedings fo the SAS Global 2018 Conference, Denver, CO. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2095-2018.pdf

SAS Micro Analytics Service
https://go.documentation.sas.com/?docsetId=masag&docsetTarget=titlepage.htm&docsetVersion=5.2&locale=en

AUTOTUNE statement
https://go.documentation.sas.com/?docsetId=casml&docsetTarget=viyaml_introcom_sect005.htm&docsetVersion=8.11&locale=en

FACTMAC procedure
https://go.documentation.sas.com/?docsetId=casml&docsetTarget=viyaml_factmac_toc.htm&docsetVersion=8.11&locale=en

ACKNOWLEDGMENTS

The author would like to thank Chris Bailey, Paul Kent, Jorge Silva, Brett Smith, Tom Weber, and Brett Wujek for their contributions to the paper. He is also grateful to <<>> for their editorial contributions

RECOMMENDED READING

- What is CI/CD?
- How to Build a Recommendation Engine Using SAS® Viya®.
- *AUTOTUNE*
- *FACTMAC*
- *MAS*

CONTACT INFORMATION

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

Jared Dean
SAS
Jared.Dean at SAS dot com