SAS® GLOBAL FORUM 2021

Paper 1136-2021

Modernizing Scenario Analysis with SAS® Viya and SAS® Visual Analytics

Arne Cordes, Volkswagen AG
Stu Sztukowski, SAS Institute Inc

ABSTRACT

Data scientists and statisticians spend months perfecting their models and presenting results. Whether it's running in SAS, Python, or R, model results need to go somewhere where someone can consume them – even the best model has little value if a decision maker can't see them. Dashboards or daily emailed reports are usually the most natural place for model results to go. To make greater use of models, decision makers need more than just the daily numbers. They need to find out how the results will change under varying conditions. This task is trivial for a data scientist: simply write a program to change the numbers and run a scenario; however, not everyone has that skillset. To quickly allow others to create scenarios, data scientists typically would place parameter estimates within a carefully crafted spreadsheet and send it to decision-makers. This inevitably results in multiple scattered versions of files that float around the company, forever to be lost in the ethos of email and shared network drives.

With SAS® Viya and SAS® Visual Analytics, all visualizations and scenarios take place from one consistent location. Not only does everyone have access to view current model results, but they can produce scenarios, score data, and run models in both SAS and open-source languages. This paper discusses methods in Viya and Visual Analytics that empower data scientists, stakeholders, and executives to run their own What-If analyses and produce better data-driven decisions.

INTRODUCTION

Visual Analytics is an easy way to display the results of statistical models in an interactive dashboard for executives and stakeholders. Everyone has one place that they go to drill into model results. Features built into Viya and Visual Analytics are available to give users the ability to run their own scenarios or score their own data and view the results within a Visual Analytics dashboard, all without needing to know any of the underlying code or model specifications. Models from SAS, R, Python, and more can all be integrated within a single underlying ecosystem, tightening version control, ensuring consistent scenario results, and allowing cross-functional teams with multiple programming language skillsets collaborate in their language of choice.

The examples shown in this paper use scoring code from a SAS model. The same basic concept still applies for open-source models: modify the data, run it through the scoring code, and output the results to the Cloud Analytics Services (CAS) server.

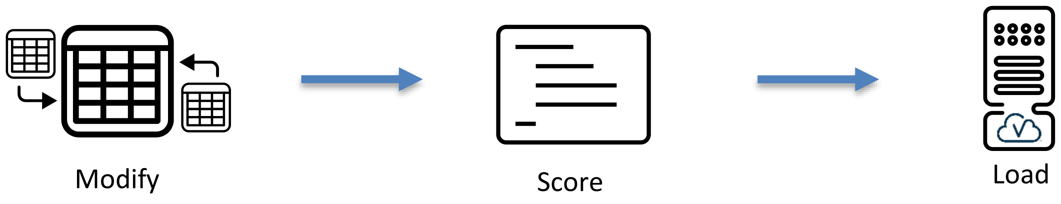


Fig. 1. Regardless of which language you choose, the steps are always the same: modify the data, score it, and load the results to CAS

For information about registering and running open-source models in Viya, see SAS4402-2020: Open Source Model Management with SAS® Model Manager and How to Execute Python or R Modules using the Open Source Code Node in SAS® Viya. Additionally, SAS `x` or `systask` commands can call Python or R interpreters directly if they are installed on the same SAS Programming Runtime Environment (SPRE) server.

This paper will go over:

- What the Job Execution Service is
- Using the Job Execution Service (JES) to run scoring code for a SAS model
- Creating a scenario dashboard that runs in Visual Analytics
- Tips for handling high-volume scenario dashboards and configurations
- Advanced topics in scenario analysis with Visual Analytics
    - The Missing Value Trick for scoring data within Visual Analytics
    - Using Data Driven Content to create scenario data
    - Modeling directly within Visual Analytics
    - Linking model results to real world results
    - Unmasking black box models

All code and examples used in this paper are available on the SAS Global Forum 2021 GitHub.

THE VIYA JOB EXECUTION SERVICE

WHAT IS THE JOB EXECUTION SERVICE?

The Viya Job Execution Service (JES) is a web-based code execution engine that runs SAS code within the SAS Programming Runtime Environment (SPRE). Jobs are flexible: they support custom HTML interfaces, prompt interfaces, applications, downloadable files, and they even integrate directly into Visual Analytics.

One way to think of the Viya Job Execution Service is to consider it a service in Viya that runs SAS code when a user accesses a URL. This URL always follows the same format:

https://viyaserver.com/SASJobExecution/?_program=/Public/MyJob

viyaserver.com is the name of your Viya Server, and */Public/MyJob* is a Viya Job named MyJob saved within the Public folder in the Viya File Service. *MyJob* contains SAS code, forms, and other instructions that control how the job runs. For the purposes of scenario analysis, this makes our lives easy: if all we need to do is access a URL to execute SAS code, we can make any process run. But what if we want to control how the job runs when we access the URL? What if we want to serve the user a form before running that code? Job Parameters are the answer.

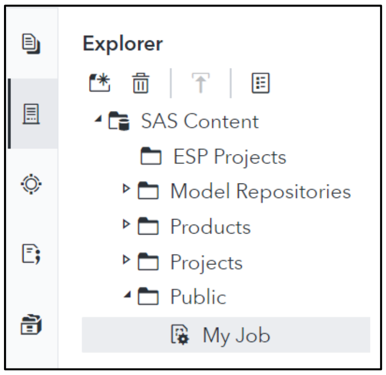


Fig. 2. Viya jobs are saved within the Viya Files Service, accessible from SAS Studio V.

JOB PARAMETERS

The Job URL holds &-separated parameters to control program flow and execution. These parameters are defined by the user within the job and are automatically translated into macro variables when the job runs. For example, the job may have a parameter named **cost** that is used to modify data in the job:

https://viyaserver.com/SASJobExecution/?_program=/Public/MyJob/**&cost=2.00**

The macro variable *cost* is accessible by the SAS program running the job:

Code	Log
`%put &cost;`	2.00

Table 1. Macro variable output from a Job input parameter.

In addition to user-defined values, the Job Execution Service has special pre-defined parameters that are used to control how the job looks and runs for the user.

Useful Pre-Defined Job Parameters

Several built-in parameters are helpful for controlling user interfaces, output types, debugging, and how the job runs. Some of the ones you will use often for scenario analysis are found below in Table 2.

Parameter	Common Arguments	Default
_action	form, execute, wait, background	form,execute
_output_type	html, ods_html5, csv, pdf, none	ods_html5
_debug	log	none

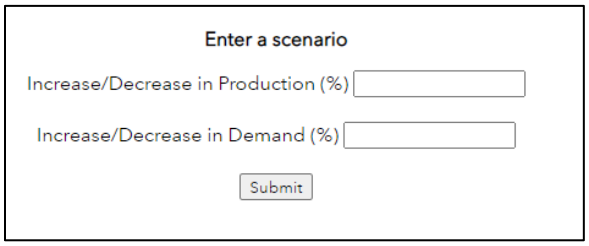
Table 2. Three of the most Common Pre-defined Job Parameters that Control Job Behavior

_Action

_Action tells the job exactly what to do when the job URL is accessed. Multiple actions can be passed by separating each with a comma. Four of the most common actions are described below: *Form*, *Execute*, *Wait*, and *Background*.

Form

The *form* action tells the job to display an input form to the user and is an excellent way to create a scenario analysis control interface. For example, a user might select a date range or modify model inputs. Viya Jobs can be associated with a form that uses either HTML or the Viya Common Task Model. *Form* is ignored if the job does not have a form.


Enter a scenario
Increase/Decrease in Production (%)
Increase/Decrease in Demand (%)
Submit

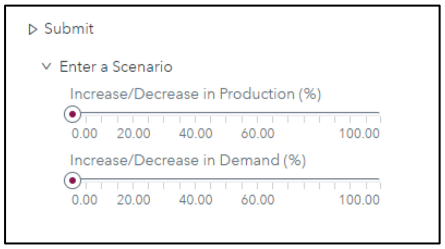


Fig. 3. The *form* action displays an interface for the user if one is associated with the job. (Left) An HTML-based form. (Right) An XML-based Viya Common Task Model form.

Execute

The *execute* action tells the job to run SAS code upon accessing the URL. If *form* is not specified, the job will run immediately. Otherwise, *form* takes precedence over *execute.*

Wait

The *wait* action displays a wait screen while the job is running. This is a convenient way to give users feedback that the job is running, requiring no custom HTML coding by the developer.

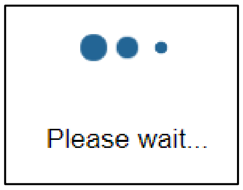


Fig. 4. The *wait* action displays an animated waiting page while the job runs.

Background

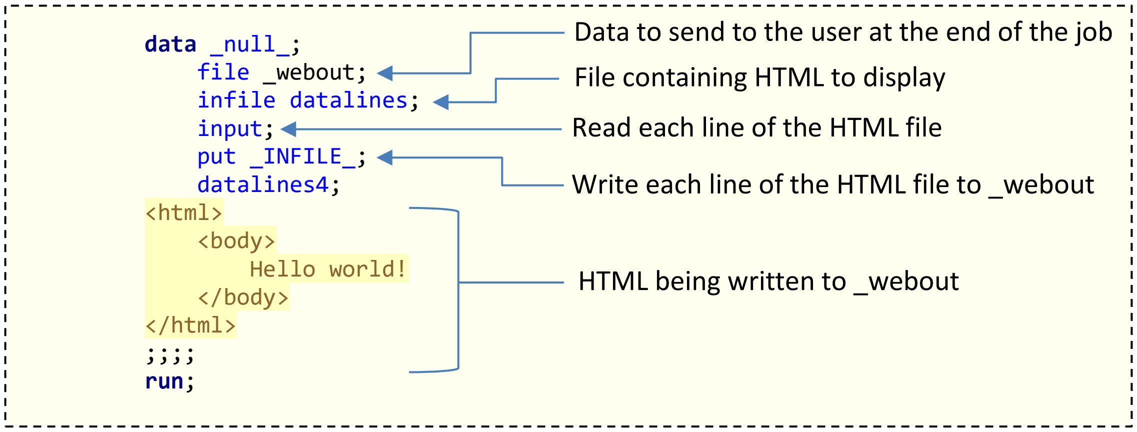
The *background* action executes a job to run in the background. The user does not need to wait for the job to complete before exiting the screen. This is helpful for complex scenarios that may take hours to run, such as optimization simulations or scenarios that modify multi-terabyte datasets. Tip: sending an email to the user through SAS is an effective way to let them know their background task has completed.

_Output_Type

_Output_Type tells the job where and how to output data at the end of a job. By default, `_output_type = ods_html5`. Any procedures that produce ODS output are written to a file reference named *_webout* and are displayed in the user's browser at the end of a job.

_webout is a special filename statement generated by the Job Execution Service that automatically delivers results to the user's browser or application. When `_output_type = html`, the job assumes anything that you write to *_webout* is HTML and will treat it as such.

You can write your own HTML via the data step and display it to the user. For example, a `data _null_` step could contain embedded HTML or reference an external HTML file.



```
data _null_;
    file _webout;
    infile datalines;
    input;
    put _INFILE_;
    datalines4;
<html>
    <body>
        Hello world!
    </body>
</html>
;;;;
run;
```

Data to send to the user at the end of the job
File containing HTML to display
Read each line of the HTML file
Write each line of the HTML file to _webout
HTML being written to _webout

Code 1. `_output_type=html` lets you display custom HTML pages within jobs.

When `_output_type=none`, the user is in full control of all output being displayed. This is most common when the results are downloaded directly to the user's desktop, such as a PDF or csv file.

_Debug

Setting `_debug=log` displays the SAS log on the screen after the job completes. This is a convenient way to view the SAS log for debugging immediately after a job is run. Omitting this parameter hides the log from the user and is desirable from a security perspective, especially if sensitive data may be displayed in the log.

CREATING A JOB TO RUN SCORING CODE FROM AN INTERFACE

EXAMPLE: FORECASTING JUMBO PRETZELS WITH SASHELP.SNACKS

Let’s say we are a large grocery store chain and jumbo pretzels are one of our most popular products. We have a forecasting model using **sashelp.snacks** that predicts the demand of jumbo pretzels sold, and store managers would like to be able to run their own scenarios to see how profit changes if the sales price or cost per bag changes. The code, dashboard, and examples below are found in the pretzel-scenario-analysis folder on GitHub.

GETTING STARTED

SAS Studio V is the primary application we will use to create and edit jobs. To work with jobs in SAS Studio V, make sure you are in the “Standard” perspective (Options → Change Perspective → Standard).

To create a new job, select New → Job → Definition. You will be greeted with a new editor. First, give your job a name and save it to a place in the Viya File Service. For this example, we will save it to “Public” and name it “MyJob.” Fig. 6 below describes the basics of the job editor.

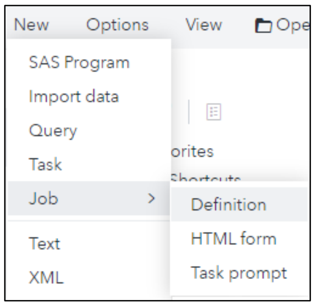
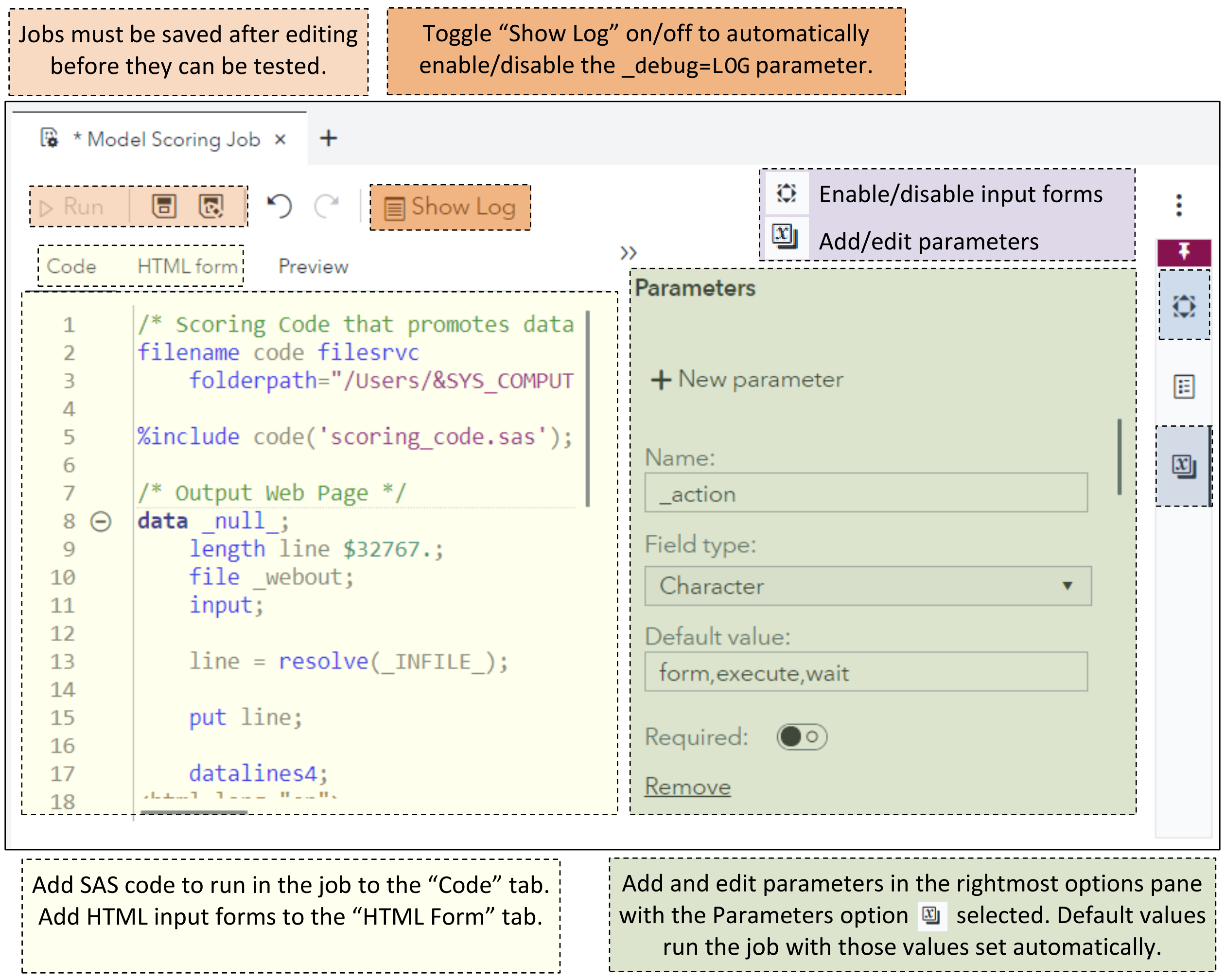


Fig. 5. Creating a new job in SAS Studio V is as easy as two clicks.



Jobs must be saved after editing before they can be tested.

Toggle “Show Log” on/off to automatically enable/disable the _debug=LOG parameter.

Enable/disable input forms

Add/edit parameters

```
/* Scoring Code that promotes data
filename code filesrvc
    folderpath="/Users/&SYS_COMPUT

%include code('scoring_code.sas');

/* Output Web Page */
data _null_;
    length line $32767.;
    file _webout;
    input;

    line = resolve(_INFILE_);

    put line;

    datalines4;
```

Name: _action

Field type: Character

Default value: form,execute,wait

Required: off

Add SAS code to run in the job to the “Code” tab. Add HTML input forms to the “HTML Form” tab.

Add and edit parameters in the rightmost options pane with the Parameters option selected. Default values run the job with those values set automatically.

Fig. 6. The job editor is a fully self-contained IDE that lets you edit and test SAS code, forms, and job parameters.

JOB PARAMETERS

First, let's set up our job parameters. Click the **Parameters option** and click + to add new parameters. By setting default values, we only need to add these parameters to our job URL when we would like to modify them.

Parameter	**Field Type**	**Default Value**	**Description**
_action	Character	form	Show a form when accessing this job
_output_type	Character	html	Display html when writing to _webout
Price	Numeric	2.49	Sale Price of pretzels with a default value
Cost	Numeric	0.25	Cost per bag of pretzels with a default value

Table 3. Our job's parameters and their default values.

START WITH THE BASICS: MODIFY, SCORE, AND LOAD

Next, we need to create our process that modifies, scores, and loads data to CAS.

Step 1: Modify

The first step is to modify our data before scoring it. Simply replace your variables of interest with your parameter values. In our case, we will set *Price* and *Cost* to the parameter values that the user will enter.

```
data pretzel_scenario_data;
    set sashelp.snacks;
    where     product = "Jumbo pretzel sticks"
          AND date BETWEEN '01JAN2003'd AND '14JAN2004'd
    ;

    /* Set future values to missing and change inputs */
    if(date > '01JAN2004'd) then do;
        call missing(QtySold);

        Price = &Price.;
        Cost  = &Cost.;
    end;
run;
```

Code 2. Modifying data with our parameters, *Price* and *Cost*, before it is scored.

Step 2: Score

The second step is to score our data. Scoring the data is exactly the same as running it in SAS Studio V or in batch mode: pass the modified data to your scoring code and let it run. In our case, we are running a forecast of jumbo pretzel demand over the next 14 days.

```
proc arima data=pretzel_scenario_data;
    identify var=QtySold crosscorr=(advertised price) noprint;
    estimate p=(1 3 7) input=(advertised price) method=ml noprint;
    forecast lead=14 id=date out=casuser.outfor(replace=yes) nooutall noprint;
run;
```

Code 3. Scoring code runs on modified data and produces updated results.

Step 3: Load

The final step is to load the data to CAS. When it comes to scenario analysis, seamlessly loading data is critical for a good user experience. We'll add a few other pieces of information to our data, such as a run timestamp, the original unmodified forecast, and original unmodified input values. We then append the table directly to the final promoted CAS table to eliminate any interruptions caused by unloading and loading the updated table.

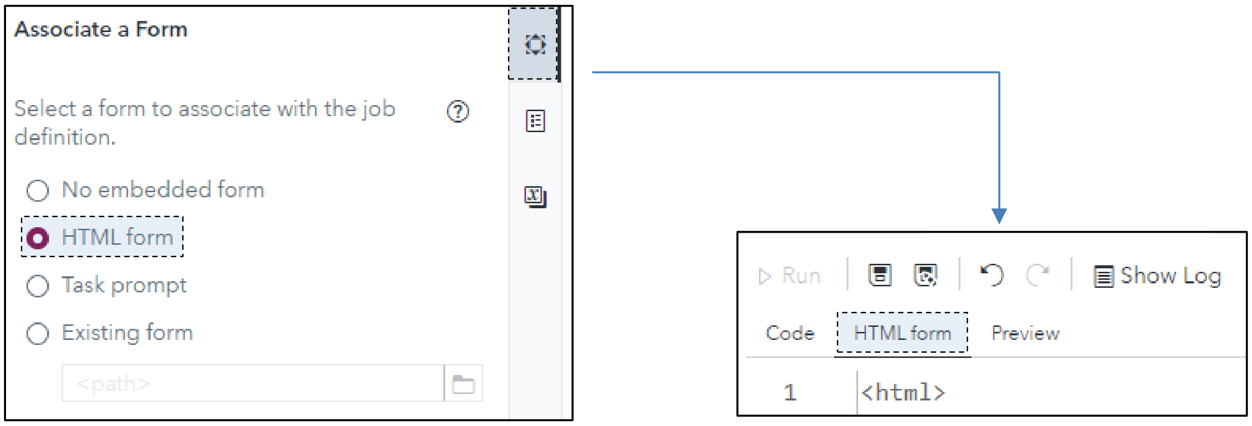
```
<other data prep code not shown>

data casuser.pretzel_scenario(append=yes);
    set casuser.pretzel_scenario_append;
run;
```

Code 4. Appending to a promoted CAS table seamlessly adds results to Visual Analytics.

Our final dataset to be used in Visual Analytics, **casuser.pretzel_scenario**, is already loaded as a promoted CAS table. You can check if the data exists by using either the `table.Exists` CAS action or `exists()` SAS function to ensure that the table is loaded as a promoted CAS table if it does not yet exist. If this is your initial scenario, manually load and promote a base scenario first.

HTML FORM INTERFACE

An HTML interface for modifying values can be as simple or complex as you'd like. If you have a strong web development background, you can create beautifully interactive interfaces for the user. For this example, we'll use a simple HTML interface to modify Price and Cost. To begin, select the **Associate a Form option** and select **HTML form** to embed an HTML form within the job. Start by adding `<html>` to your editor.


Associate a Form

Select a form to associate with the job definition.

No embedded form [✓ HTML form] Task prompt Existing form

```
<html>
```

Fig. 7. Embedding an HTML form adds the "HTML form" tab where a web page can be built.

Step 1: <Style>

Cascading Style Sheets (CSS) let us apply styles to HTML while keeping all of the settings in one place. We want to add a few styles to help make our interface feel like it's in Visual Analytics and improve spacing in our input form.

```
<head>
    <style type="text/css">
        @font-face {
            font-family: AvenirNext;
            src: url("/SASJobExecution/images/AvenirNextforSAS.woff") format("woff");
        }

        body {
            font-family: AvenirNext, Helvetica, Arial, sans-serif;
            text-rendering: optimizeLegibility;
            -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
            text-align: center
        }

        .wrapper {
            padding-bottom: 10px;
        }
    </style>
</head>
```

Code 5. Some basic CSS style elements can spruce up your interface.

Step 2: <Body>

The body of the HTML is where all the work happens. This is where we are making our form that creates parameters and calls our job to execute. Forms automatically build URLs for us, adding &'s between each parameter we add. Our HTML form needs five total parameters:

Required for most jobs

Name	Value	Type	Description
_action	execute,wait	Hidden	Run the job and display a waiting page
_program	$PROGRAM$	Hidden	Key word to run this specific program we are in
_csrf	$CSRF$	Hidden	Key word to add a Cross-Site Request Forgery Token. This is a security feature that prevents unauthorized access. Additional information.
Price		Text	Sale Price of pretzels
Cost		Text	Cost per bag of pretzels

Table 4. HTML form parameters translate to job parameters and are added to the job URL.

Our example uses single text inputs, but these are not the only types of parameters that can be passed to a job. Check out this example on how to use check boxes to pass multi-valued parameters and use the PARAM_LIST macro to make them easier to work with.

```
<body>
    <form action="/SASJobExecution/" enctype="multipart/form-data" method="post">
        <input type="hidden" name="_program" value="$PROGRAM$"/>
        <input type="hidden" name="_action" value="execute,wait"/>
        <input type="hidden" name="_csrf" value="$CSRF$"/>

        <p><b>Enter a Pretzel Scenario</b></p>

        <div class="wrapper">
            <label>Price (USD) <input type="text" name="price" placeholder="2.49"/></label>
        </div>

        <div class="wrapper">
            <label>Cost (USD) <input type="text" name="cost" placeholder="0.25"/></label>
        </div>

        <input type="submit" id="submit" value="Submit"/>

    </form>
</body>
```

Code 6. <form> is the main interface where users modify job parameters.

Let’s break down these chunks of code.

1. `<form action="/SASJobExecution/" enctype="multipart/form-data" method="post">`
    - When the user submits the form, build a URL that starts as follows: https://viyaserver.com**/SASJobExecution/**

2. `<input type="hidden" name="…">`
    - Include these parameters in the URL, but hide them from the user in the form: https://viyaserver.com/SASJobExecution**/?_program=/Public/MyJob&_action=…**

3. `<label>Price (USD) <input type="text" name="price" placeholder="2.49"/></label>`
    - Add a text input box whose parameter name is “price,” and use 2.49 as a default.

4. `<label>Cost (USD) <input type="text" name="cost" placeholder="0.25"/></label>`
    - Add a text input box whose parameter name is “cost,” and use 0.25 as a default.

5. `<input type="submit" id="submit" value="Submit"/>`
    - Create a button that submits the form when users click it.

Note that all of our HTML parameter names, denoted by `name=`, are the same as our job parameters. The name is passed as a parameter into the URL built by the form. Single inputs are not the only way to pass parameters.

A Note on _program

You may be wondering why our form calls the job that we are already in. Recall that our default action for the job is **`form`**. When we access this job, it brings up the HTML form without executing any SAS code. We need to call it a second time with the **`execute`** action in order to run the job with our modified parameters. Since SAS knows what job we are in, the **`$PROGRAM$`** keyword is used to automatically add this job’s location to the `_program` parameter without having to retype the full job location. Instead of hosting a separate web page and pointing it to a job, a single process controls both. This makes jobs more generalizable so that they can be easily used across multiple projects and environments.

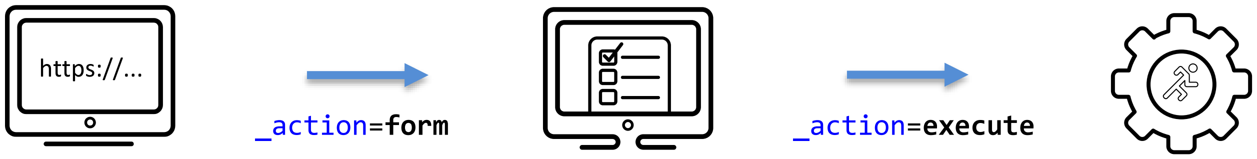


Fig. 8. Viya Jobs have dual functions: (1) serve the user a form, and (2) execute SAS code.

Step 3: Test

End your HTML with `</html>`, save your job, and click "Run" to review and test your form. You can confirm if your process is successful by checking your output datasets. Click the "Show Log" button for additional information on what is happening in the background.

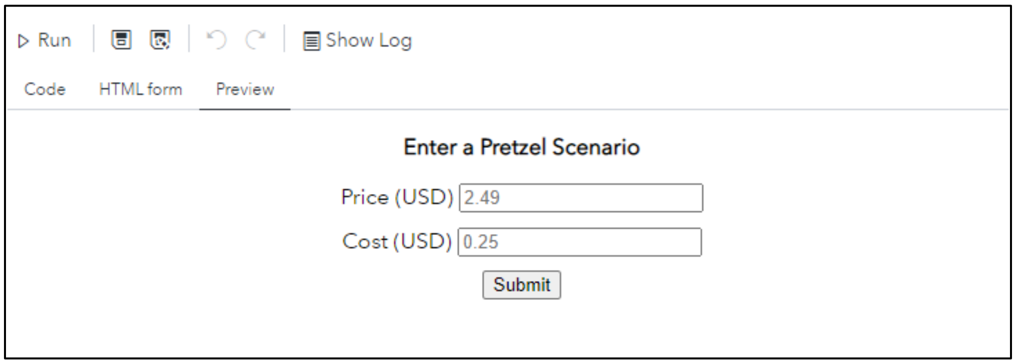


Fig. 9. SAS Studio V has tools to test, edit, and run your job from one place.

JOB RESULTS AND FEEDBACK

When we run our job, the final output screen is blank. The user might not know if the job was successful. Letting the user know that their action was successful or not successful will reduce questions that may come your way.

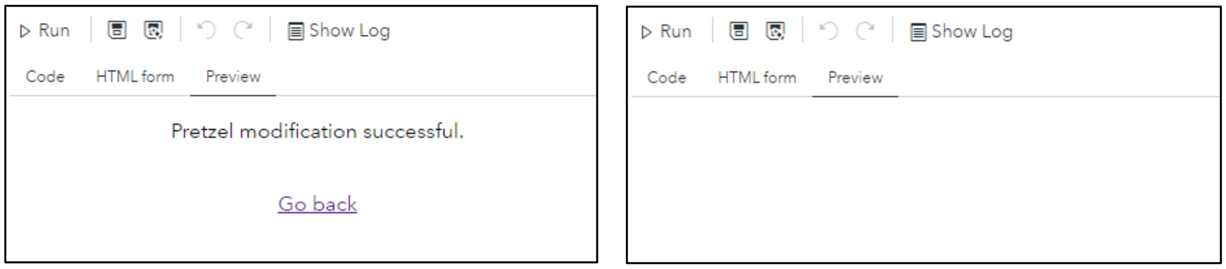


Fig. 10. Did the job run successfully?
A result page (Left) makes a big difference in usability compared to none (Right).

We'd like to give the user feedback that the job is complete. Recall that we set our **`_output_type`** parameter to **html**. Our job is expecting some HTML output for the user, but we have yet to make it. Let's add a small webpage that lets the user know the job was successful. Click the "Code" tab and add the following code after your scoring code:

```
data _null_;
    length line $32767.;
    file _webout;
    input;
    line = resolve(_infile_);
    put line;
    datalines4;
<html>
    <body>
        <p>Pretzel modification successful.</p>
        <br/>
        <a href="/SASJobExecution/?_program=&_program">Go back</a>
    </body>
</html>
;;;;
run;
```

Data to send to the user at the end of the job

Read each line of the HTML file

Treat &-prefixed text as macro variables

Write each line of the HTML file to _webout

Job-generated macro variable for the location of this job

Code 7. Result web pages are written to the fileref _*webout*.

This code writes HTML to `_webout`. We have already told the job to expect HTML, so the job will know what to do with it. While we do not need to write our HTML within a `datalines4` statement, it is a convenient way of adding HTML without storing a separate file on the server – especially for such a small amount of HTML.

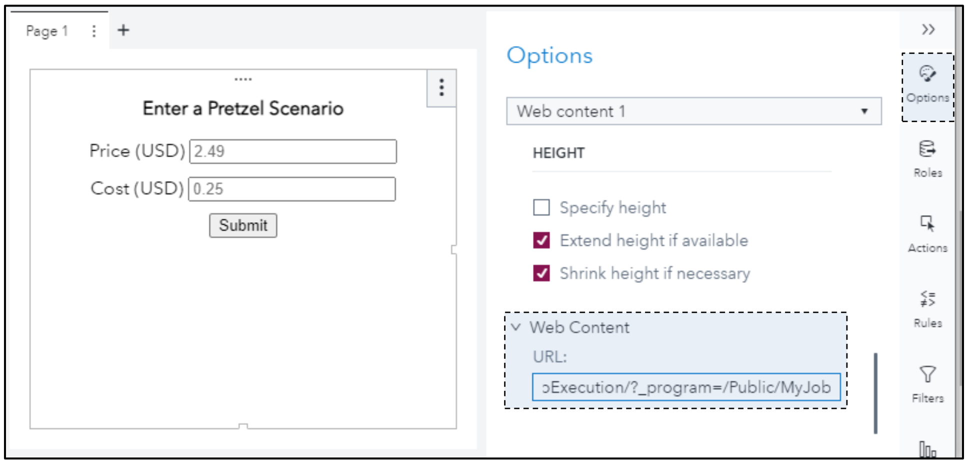
SCENARIO ANALYSIS IN VISUAL ANALYTICS

Now that we have our job running successfully, the hard part is over: all we need to do is add the job to a Visual Analytics dashboard and build a dashboard around the final output dataset, *casuser.pretzel_scenario*. The Web Content Object lets you add websites as inline-frames to your dashboard.

ADDING A JOB TO A DASHBOARD

To configure your web content object, do the following:

1. Drag the Web Content Object into the report
2. Select Options and scroll to "Web Content" option at the bottom
3. Enter the URL of your job. Replace "viyaserver.com" with your Viya server's name. https://viyaserver.com/SASJobExecution/?_program=/Public/MyJob


Enter a Pretzel Scenario

Price (USD) 2.49

Cost (USD) 0.25

Options

Web content 1

HEIGHT

☐ Specify height

☑ Extend height if available

☑ Shrink height if necessary

Web Content

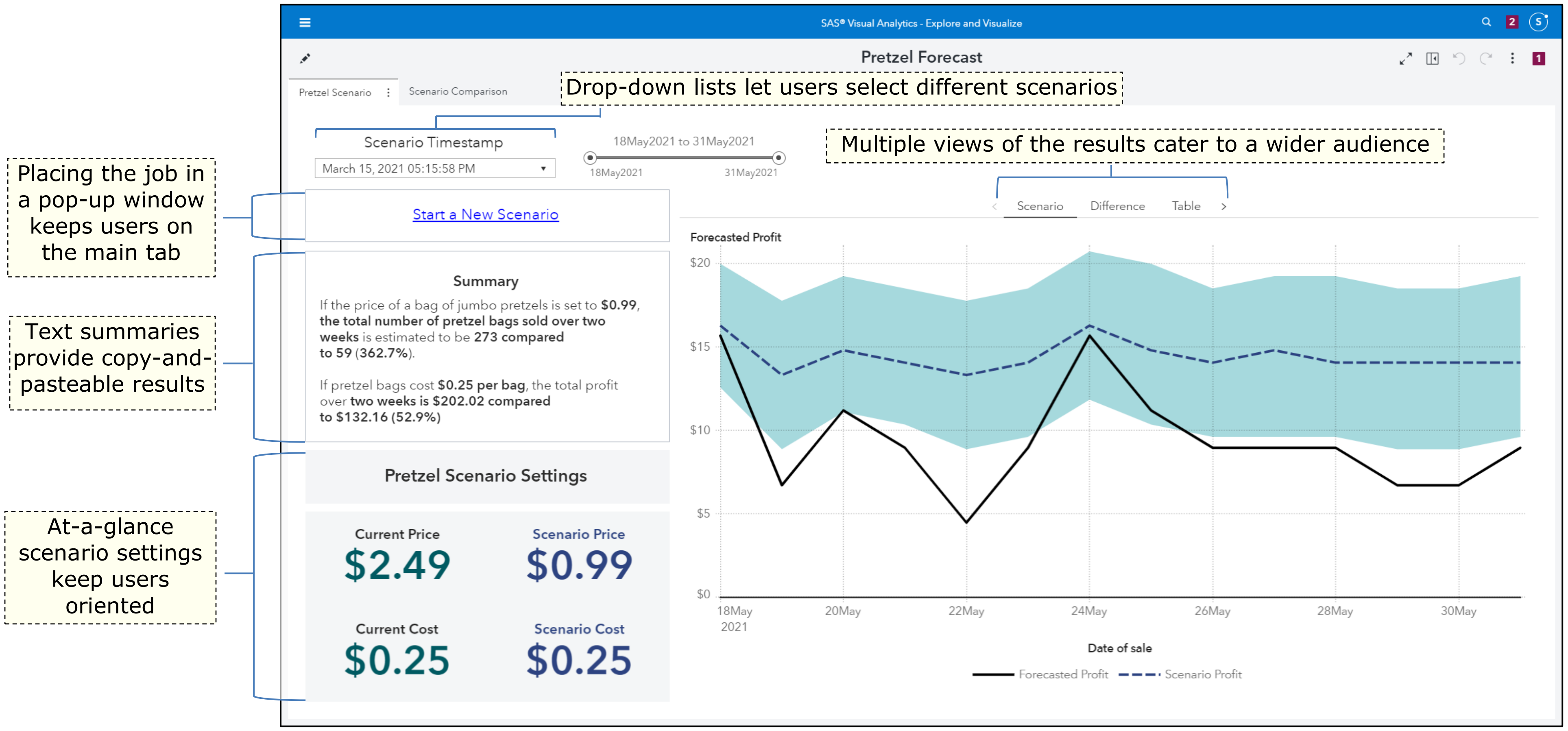
URL: ɔExecution/?_program=/Public/MyJob

Fig. 11. The Web Content object in Visual Analytics is used to display Viya Job Forms.

SCENARIO ANALYSIS DASHBOARD TIPS

There are numerous interfaces and styles to consider when making a Visual Analytics interface that all depend on the type of analysis being done and who your audience is. Below are some general guidelines.

- Set the dashboard to check for new data every 5 seconds. Users will be able to select their scenario once it completes.
- Include your original, unmodified values. Users should be able to compare their scenario to a baseline.
- Add multiple views: graphs, tables, etc. Try to accommodate multiple ways people like to read and interpret data.
- Consider adding a comparison tab that allows users to compare all scenarios with each other.



Drop-down lists let users select different scenarios

Multiple views of the results cater to a wider audience

Placing the job in a pop-up window keeps users on the main tab

Text summaries provide copy-and-pasteable results

At-a-glance scenario settings keep users oriented

Summary

If the price of a bag of jumbo pretzels is set to **$0.99**, **the total number of pretzel bags sold over two weeks** is estimated to be **273 compared to 59 (362.7%)**.

If pretzel bags cost **$0.25 per bag**, the total profit over **two weeks is $202.02 compared to $132.16 (52.9%)**

Current Price	Scenario Price
$2.49	$0.99
Current Cost	**Scenario Cost**
$0.25	$0.25

Fig. 12. An example of a scenario analysis dashboard. Focus on the user experience and keep your audience in mind.

HIGH-VOLUME DASHBOARDS

When a few users have access to run scenario analyses, it is unlikely that they will both try to simultaneously run a scenario at the same time and lock one another from writing to the same table in CAS. The odds of this increases with a higher user base, making for a frustrating experience that results in a dashboard filled full of many users' scenarios. Using a few simple tricks to check if someone is running a scenario and a built-in Viya tool to control row-level access, you can prevent accidental collisions and ensure scenarios are seen only by specific users.

PREVENTING COLLISIONS

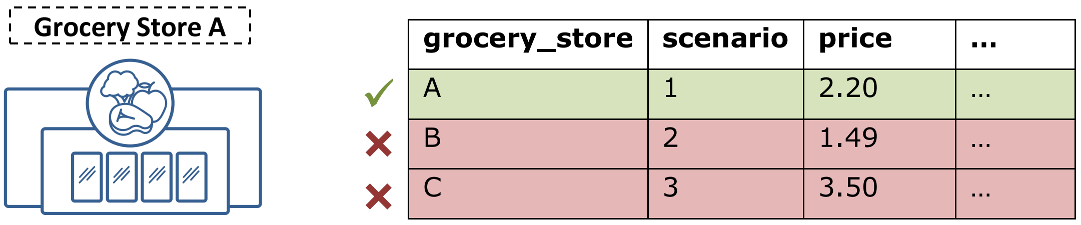
One solution to prevent collisions between multiple users is to add a blank dataset to a common location that all users can access. The job attempts to lock this file when it runs. If it's successful, the job continues and unlocks the file when it completes. If the lock isn't successful, the job waits a number of seconds and attempts to lock this file again. Too many failed attempts will display a result page to the user that another user is currently running a scenario, and that they will need to wait and try again another time.

```
%macro runJob(maxtries=10, wait=10);
    %let flag_timeout = 0;
    %let tries        = 0;

    %if(%sysfunc(exist(lib.job_lockfile)) = 0) %then %do;
        data lib.job_lockfile;
        run;
    %end;

    /* Keep trying to lock the lockfile until max attempts are reached */
    %do %until(&syslckrc. LE 0 OR &tries. GE &maxTries.);

        /* Count total number of tries */
        %let tries = %eval(&tries. + 1);

        /* Try to lock the lockfile */
        lock lib.job_lockfile nomsg;

        /* If it's locked and we're under our max tries, wait, then try again */
        %if(&syslckrc. > 0 AND &tries. < &maxTries) %then %let rc = %sysfunc(sleep(&wait., 1));
            %else %if(&syslckrc. > 0) %then %let flag_timeout = 1;
    %end;

    %if(&flag_timeout. = 0) %then %do;
       /* Continue and run the job */
    %end;
        %else %do;
            /* Display timeout page */
        %end;
%mend;
```

Code 8. Checking if a file is locked keeps users from colliding with each other.

RESTRICTING RESULTS

In our example, multiple grocery stores are using a single scenario dashboard. We may want to restrict our dashboard to only show results by store. Row-level security in Viya automatically applies filters to a table so that only users that match certain criteria can view rows in a dataset. Check out David Stern's SAS Communities Article and video for a detailed explanation and examples of how to apply row-level security on CAS tables based on user attributes.

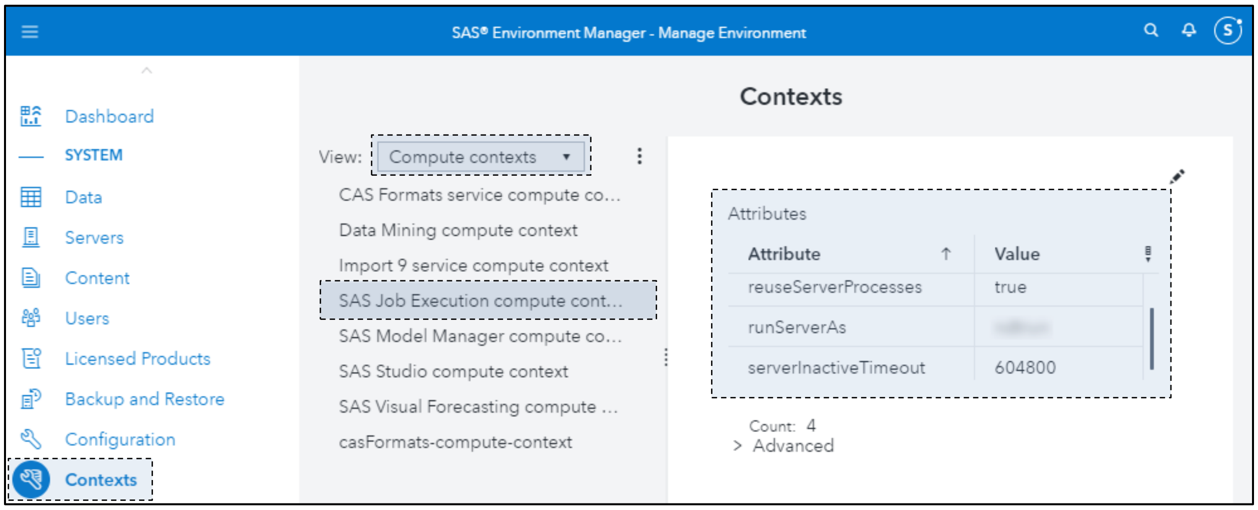


	grocery_store	scenario	price	...
✓	A	1	2.20	...
✗	B	2	1.49	...
✗	C	3	3.50	...

Fig. 13. Row-level security filters rows of a table based on user attributes.

JOB EXECUTION SERVICE CONFIGURATION TIPS

By default, Viya Jobs run under your user ID and a compute process is created each time the job is invoked. Permissions issues may occur when many different users are running jobs. This can be resolved by running all jobs under a shared user ID. Additionally, jobs can run faster by reusing previous compute sessions. These are configured by your administrator in minutes and take effect immediately. For information on how to configure this, see Server Contexts: How To.



Contexts

View: Compute contexts

- CAS Formats service compute co...
- Data Mining compute context
- Import 9 service compute context
- SAS Job Execution compute cont...
- SAS Model Manager compute co...
- SAS Studio compute context
- SAS Visual Forecasting compute ...
- casFormats-compute-context

Attributes

Attribute	Value
reuseServerProcesses	true
runServerAs	[redacted]
serverInactiveTimeout	604800

Count: 4
> Advanced

Fig. 14. Configuring the SAS Job Execution Compute Context to run under a shared user and to reuse previous job compute processes reduces permissions issues and improves performance.

Note that when running under a shared user ID, the macro variable `&sysuserid` will resolve to the shared user ID. To get the user ID that invoked the job, use the macro variable `&sys_compute_session_owner` instead.

ADVANCED SCENARIO TOPICS IN VISUAL ANALYTICS

After setting the conceptual groundwork for job execution and seeing how the pieces fit together in a Visual Analytics Report, let us look at some advanced scenario analysis topics using a real world example with Volkswagen Financial Services and Residual Car Value.

Decision makers rarely share the data analysts' enthusiasm for how they solved a given problem analytically. They are more practical and they want the analytical model to create benefit for the business. This reads often as scoring a model. However, they need to build trust in the model and assess its response to changing business scenarios managers can control and understand. The sweet spot for this challenge is Visual Analytics, which grants Data scientists and decision makers a rich experience in a collaborative manner.

The code, examples, and data used in this example are found in the residual-car-analysis folder on GitHub.

EXAMPLE: RESIDUAL CAR VALUE

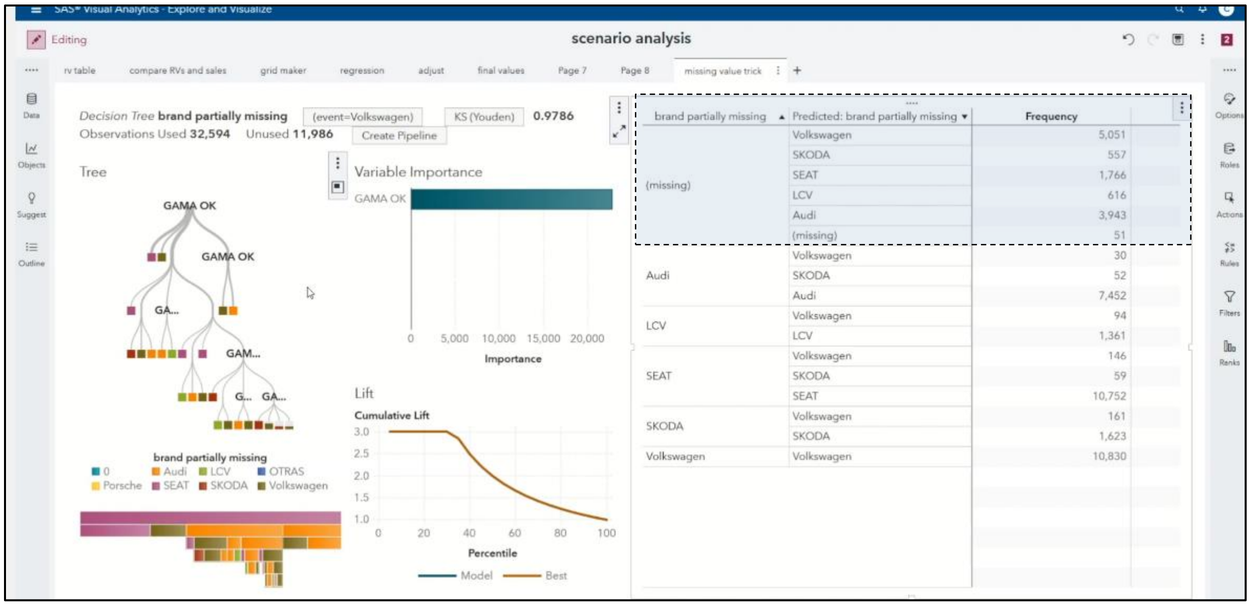
Volkswagen Financial Services offers long-term car rentals with many different optional services that cost a monthly fee, such as insurance and maintenance plans. The main cost driver for the monthly fee is the decrease in value for the car. In this context, residual value is the price the market is willing to pay for a used car having a specific set of features like miles driven, age, optional equipment, and more. The better a car can retain its residual value the lower the monthly installment will be.

It is crucial to accurately estimate the residual value. If we overestimate a car's prospects of remarketing it as a used vehicle, we will have to write off losses when the customer returns the car and the sales price is below its accounted residual value. The losses inflate as many customers chose this car because of its falsely calculated "competitive" monthly rate, a clear example of negative selection. On the other hand, a residual value, which is too conservative, will fail to attract customers, as the monthly rate is out-of-market.

THE MISSING VALUE TRICK: SCORING DIRECTLY IN VISUAL ANALYTICS

Most algorithms available in Visual Analytics, Visual Statistics, and Visual Data Mining and Machine Learning can derive a score once the objective function has given a solution. It is as easy as right clicking the model pane menu and selecting the option "Derive predicted…".

The procedure creates estimates (and residuals) alongside the target variable's value in those observations we used for training the model, and it even calculates a score for those observations where the target value is missing.



brand partially missing	Predicted: brand partially missing	Frequency
(missing)	Volkswagen	5,051
	SKODA	557
	SEAT	1,766
	LCV	616
	Audi	3,943
	(missing)	51
Audi	Volkswagen	30
	SKODA	52
	Audi	7,452
LCV	Volkswagen	94
	LCV	1,361
SEAT	Volkswagen	146
	SKODA	59
	SEAT	10,752
SKODA	Volkswagen	161
	SKODA	1,623
Volkswagen	Volkswagen	10,830

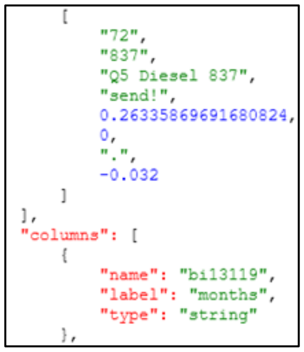
Fig. 15. Missing values of the target variable receive a score when deriving predicted values.

The missing value trick uses this behavior. When applied correctly it prevents you from writing dedicated scoring code thus offering an elegant way to do the scoring directly in Visual Analytics. Managers are much more familiar with translating a score to a representative car instead of scrolling through the real world messy data set.

The idea is to append a data grid with relevant combinations of the model variables to the original data set. For more information on this, check out Rick Wicklin's blog post on scoring a regression model using the Missing Value Trick.

USING DATA DRIVEN CONTENT TO CREATE SCENARIO DATA IN VISUAL ANALYTICS

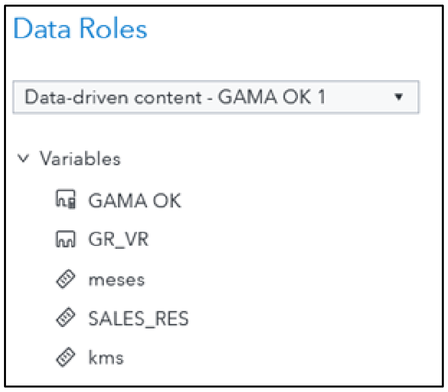
Instead of users creating scenario data within Excel and uploading it to Visual Analytics, we can leverage Data Driven Content and the Job Execution Service to create new scenario data using controls within Visual Analytics.



```
        [
            "72",
            "837",
            "Q5 Diesel 837",
            "send!",
            0.26335869691680824,
            0,
            ".",
            -0.032
        ]
    ],
    "columns": [
        {
            "name": "bi13119",
            "label": "months",
            "type": "string"
        },
```

Fig. 16. The Data Driven Content Viewer shows how your data is being propagated

The URL for your Data Driven Content is your server address followed by the specific path to where your JES code resides. The Data Driven Content object passes all the required variables and parameters to the Job Execution Service where a JSON object is written to a data set. Custom SAS code then appends the new scenario data with missing values to the original data. The Data Driven Content Data Viewer is a great resource to understand how variables and parameters are propagated.



Data Roles

Data-driven content - GAMA OK 1

Variables
- GAMA OK
- GR_VR
- meses
- SALES_RES
- kms

Fig. 17. Data Driven Content can pass Visual Analytics variable values to the Job Execution Service.

For more information about using Data Driven Content, check out [Renato Lupi's SAS Communities post on Using parameters with Data-Driven Content in SAS Visual Analytics]().

Creating Scenario Data

Decision makers often require filtering the training data. With some instructions added to the report, decision-makers are empowered to select the variables that they believe should enter the model. Letting the subject matter expert pre-select variables based on their expertise helps buy-in their support for the model at an early stage.

Although the guideline and the how-to are best covered by Renato Lupi's blog, use the following basic steps:

1. Drag the Data Driven Content into the report
2. Select Options and scroll to "Web Content" option at the bottom
3. Enter the URL of your job containing your code (here). Replace "viyaserver.com" with your Viya server's name.
   https://viyaserver.com/SASJobExecution/?_program=/Public/MyDDCJob
4. Create the required parameters that the underlying job execution expects
5. Make sure the Data Driven Content includes the parameters by adding a fake advanced filter that references to these parameters
6. Load the output table once to the report, after it runs successfully for the first time it updates automatically

MODELING AND RUNNING SCENARIOS DIRECTLY WITHIN VISUAL ANALYTICS

The output table from the previous step feeds the analytical model. Note that that not only the real observations receive a score but also the new scenario data with missing values for the target variable.

Through report controls, we let the user choose specific models at dedicated mileage combinations and make commercial changes. Users visually assess how the commercial changes affect the resulting residual values and how they compare against the current tariffs. Normally decision makers meet several times until they close the final residual values. We use a time stamp to create a history of how the residual values change between these meetings.

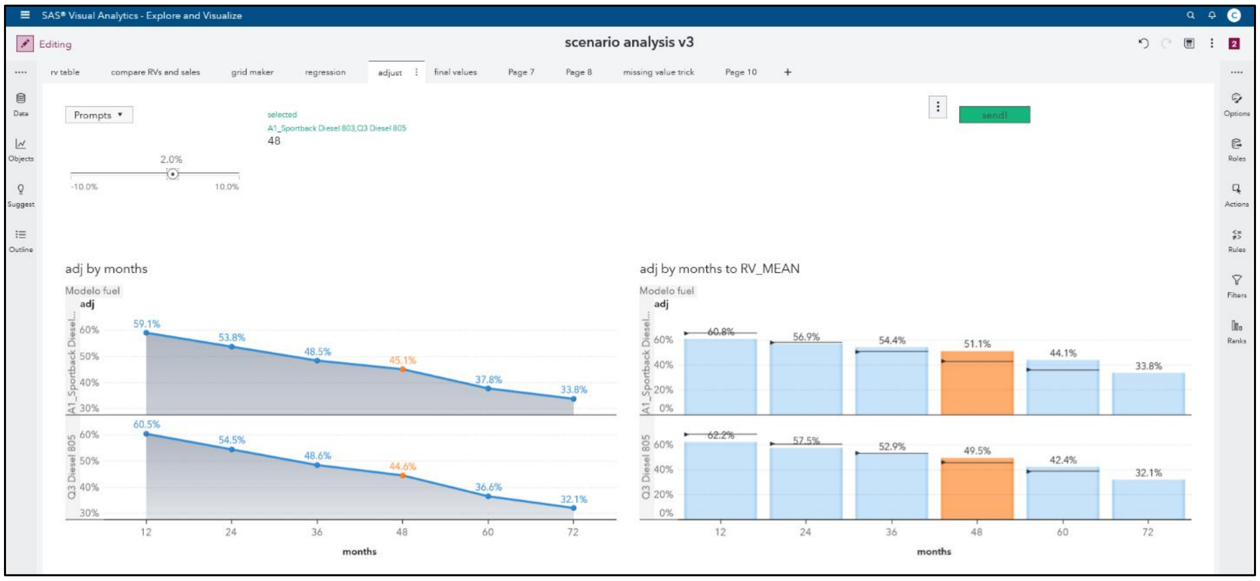


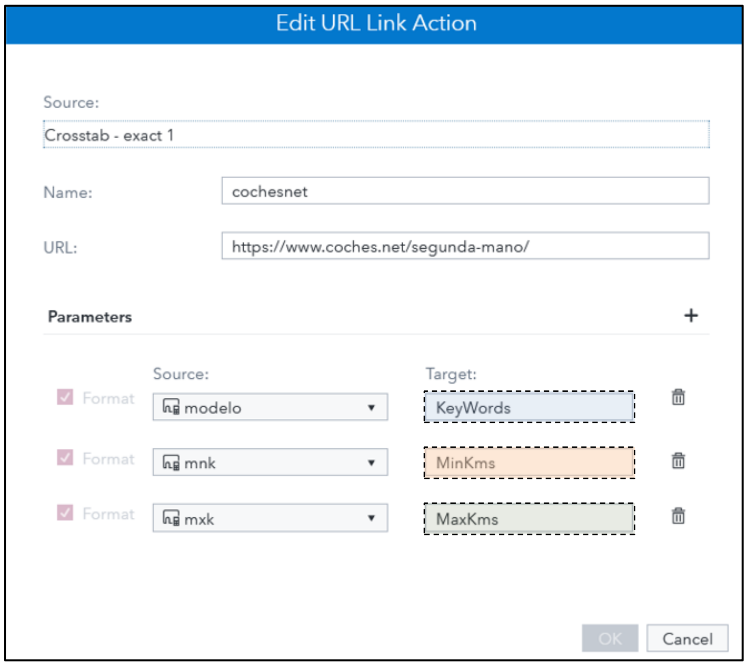
Fig. 18. Decision makers select car models and terms that receive a commercial adjustment.

Verifying Model Results with Real World Data

The sales data is restricted to the car models of the Volkswagen group brands sold in the regime of renting & leasing. That means that it is only a subset of the entire used car market. Internet platforms act as matchmaker between offer and demand of used cars and serve a reality check for the residual values that our analytical model estimates. Linking the report to a popular car sales website adds a reality check to the validity of a model, adding an easy way to improve model buy-in.

A Crosstab in Visual Analytics can pass character values as parameters to a URL. To link a URL and pass parameters, perform the following:

1. Determine how the URL resolves the filters you enter at the website by investigating the URL link. Any URL parameters prefixed by ? or &.
2. Drag the Crosstab into the report and add all variables of interest
3. Convert any numerical values to character values
4. Add a URL link to the Crosstab
5. Add parameters according to the structure identified in (1) and click “OK”
6. Click on the crosstab will open a new browser window with the internet platform having filtered for the row of the Crosstab
7. Review the price the market calls for comparable car


Edit URL Link Action

Source:
Crosstab - exact 1

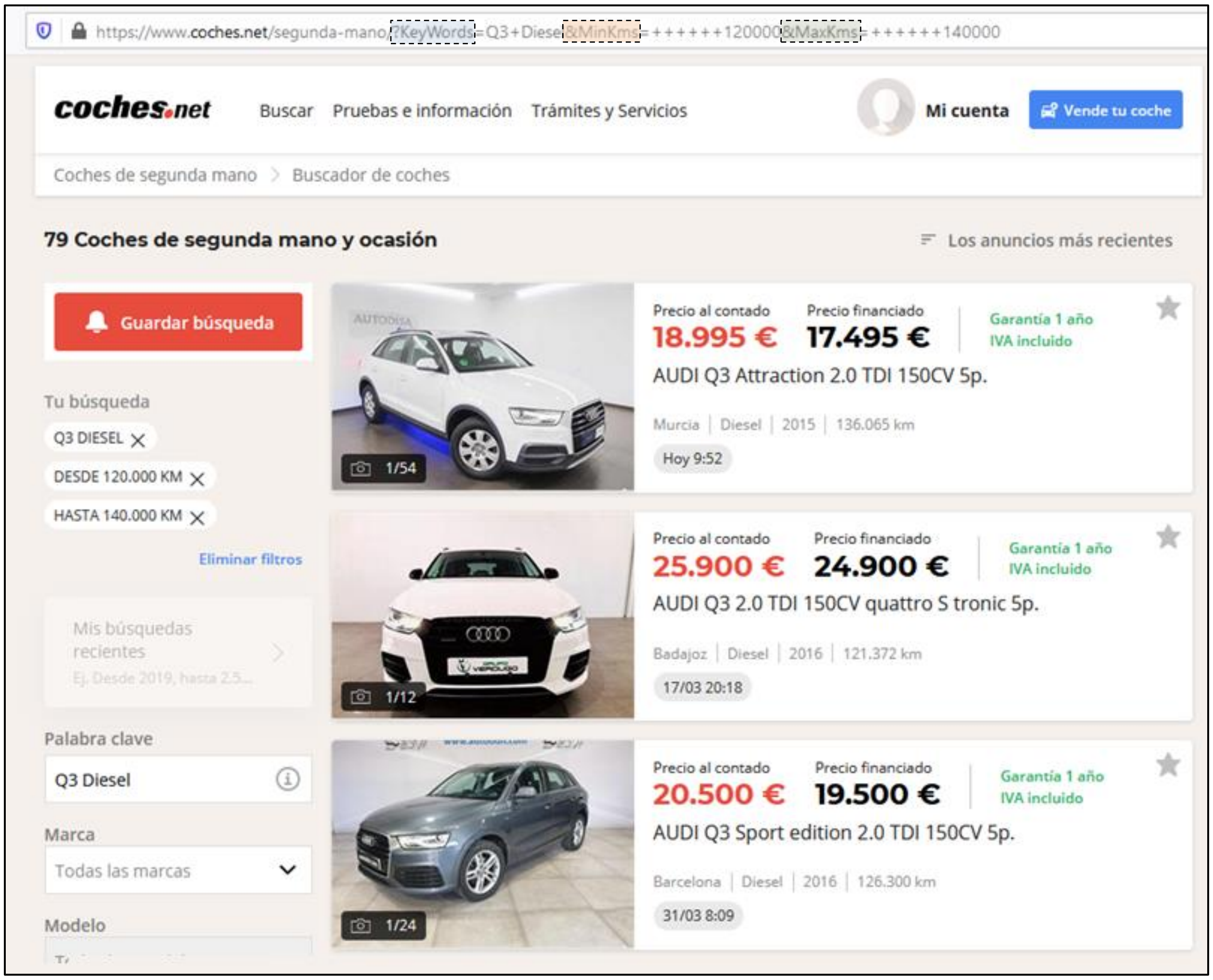
Name: cochesnet

URL: https://www.coches.net/segunda-mano/

Parameters

	Source:	Target:
✓ Format	modelo	KeyWords
✓ Format	mnk	MinKms
✓ Format	mxk	MaxKms

Fig. 19. Set target URL parameters in Visual Analytics to the way the URL expects them.


https://www.coches.net/segunda-mano/?KeyWords=Q3+Diesel&MinKms=++++++120000&MaxKms=++++++140000

79 Coches de segunda mano y ocasión

Tu búsqueda: Q3 DIESEL; DESDE 120.000 KM; HASTA 140.000 KM

Palabra clave: Q3 Diesel

Marca: Todas las marcas

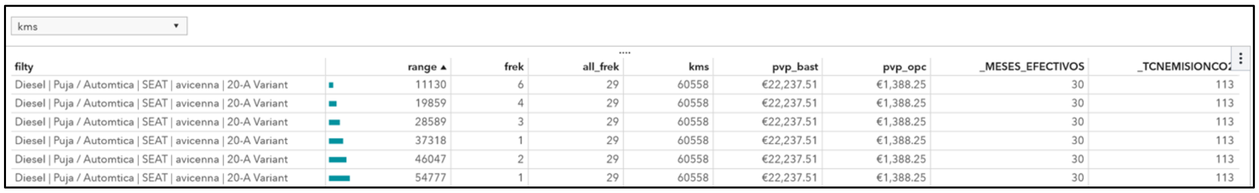
Precio al contado	Precio financiado		Modelo	Datos	Fecha
18.995 €	17.495 €	Garantía 1 año, IVA incluido	AUDI Q3 Attraction 2.0 TDI 150CV 5p.	Murcia | Diesel | 2015 | 136.065 km	Hoy 9:52
25.900 €	24.900 €	Garantía 1 año, IVA incluido	AUDI Q3 2.0 TDI 150CV quattro S tronic 5p.	Badajoz | Diesel | 2016 | 121.372 km	17/03 20:18
20.500 €	19.500 €	Garantía 1 año, IVA incluido	AUDI Q3 Sport edition 2.0 TDI 150CV 5p.	Barcelona | Diesel | 2016 | 126.300 km	31/03 8:09

Fig. 20. Double-clicking a row in the Crosstab automatically searches a website with the selected parameters. Users can compare predicted results with real-world data.

UNMASKING BLACK-BOX ALGORITHMS

Colleagues outside the analytical field may feel uncomfortable when dealing with advanced algorithms. They need to understand what the model does, but they also demand the best-performing model. Top-performing models often lead to black-box algorithms like gradient boosting. Variable importance charts, ROC curves, or misclassification rates improve user confidence, but they do not answer all questions about model behavior. Instead, we can create our own interpretability charts and show them in Visual Analytics.

In order to have a representative and relatable grid of points, we limit the scope to the top 80 as of Pareto combinations of the class variables that from the model. Then we create 20 (adjustable) evenly spaced values for each of the numerical variables, starting at its min and ending at its max, and calling a mean value for all other numeric values.

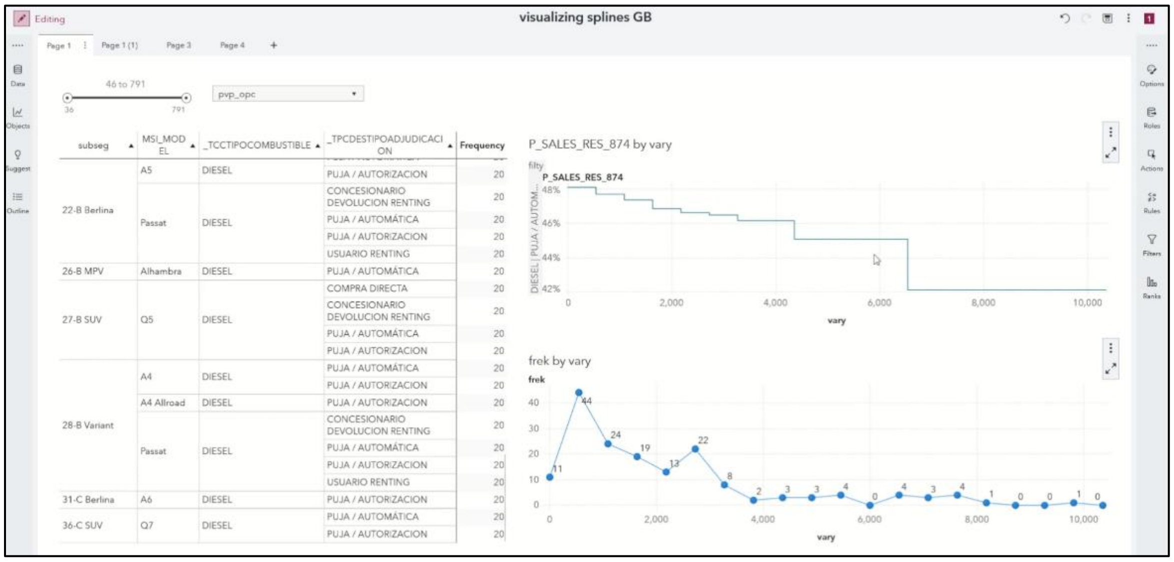


kms

filty		range ▲	frek	all_frek	kms	pvp_bast	pvp_opc	_MESES_EFECTIVOS	_TCNEMISIONCO2
Diesel | Puja / Automtica | SEAT | avicenna | 20-A Variant		11130	6	29	60558	€22,237.51	€1,388.25	30	113
Diesel | Puja / Automtica | SEAT | avicenna | 20-A Variant		19859	4	29	60558	€22,237.51	€1,388.25	30	113
Diesel | Puja / Automtica | SEAT | avicenna | 20-A Variant		28589	3	29	60558	€22,237.51	€1,388.25	30	113
Diesel | Puja / Automtica | SEAT | avicenna | 20-A Variant		37318	1	29	60558	€22,237.51	€1,388.25	30	113
Diesel | Puja / Automtica | SEAT | avicenna | 20-A Variant		46047	2	29	60558	€22,237.51	€1,388.25	30	113
Diesel | Puja / Automtica | SEAT | avicenna | 20-A Variant		54777	1	29	60558	€22,237.51	€1,388.25	30	113

Fig. 21. Evenly spaced numeric variables at relevant combinations of categorical variables.

This technique guarantees that the derived score happens at "close-to-reality" combinations. We also apply a ceteris paribus approach as only the variable of interest changes and affects the score. After creating the grid, we derive a score via code (here) and use the scored table for our report.



subseg ▲	MSI_MODEL ▲	_TCCTIPOCOMBUSTIBLE ▲	_TPCDESTIPOADJUDICACION ▲	Frequency
22-B Berlina	A5	DIESEL	PUJA / AUTORIZACION	20
	Passat	DIESEL	CONCESIONARIO DEVOLUCION RENTING	20
			PUJA / AUTOMÁTICA	20
			PUJA / AUTORIZACION	20
			USUARIO RENTING	20
26-B MPV	Alhambra	DIESEL	PUJA / AUTOMÁTICA	20
27-B SUV	Q5	DIESEL	COMPRA DIRECTA	20
			CONCESIONARIO DEVOLUCION RENTING	20
			PUJA / AUTOMÁTICA	20
			PUJA / AUTORIZACION	20
28-B Variant	A4	DIESEL	PUJA / AUTOMÁTICA	20
			PUJA / AUTORIZACION	20
	A4 Allroad	DIESEL	PUJA / AUTORIZACION	20
	Passat	DIESEL	CONCESIONARIO DEVOLUCION RENTING	20
			PUJA / AUTOMÁTICA	20
			PUJA / AUTORIZACION	20
			USUARIO RENTING	20
31-C Berlina	A6	DIESEL	PUJA / AUTORIZACION	20
36-C SUV	Q7	DIESEL	PUJA / AUTOMÁTICA	20
			PUJA / AUTORIZACION	20

Fig. 22. Visualize "how" the algorithm scores along the data range.

This way we isolate the effect of the numerical variable and visualize how the algorithms changes its prediction as values increase. It helps to identify critical points where the score function jumps suddenly. For example, we can identify when there is "too-much" optional equipment that hamper the prospect of the residual value.

CONCLUSION

Key decision makers have a tough job – they may need to make multi-million dollar decisions based on a number of factors, and your model may be at the forefront of that of that decision. SAS Viya has built-in tools to runs SAS models, open source models, and modern interfaces with little to no HTML coding experience needed. By combining the SAS Job Execution Service with Viya and Visual Analytics, you empower everyone to make data-driven decisions from sophisticated models at the click of a button all from one location.

REFERENCES

Bizoux, Xavier. "Data Entry in SAS Visual Analytics 8.3: Part 1, The Basics." SAS Communities, SAS Institute, 20 Aug. 2019, https://communities.sas.com/t5/SAS-Communities-Library/Data-Entry-in-SAS-Visual-Analytics-8-3-Part-1-The-basics/ta-p/579076?nobounce.

Duling, David. "The Aftermath What Happens After You Deploy Your Models and Decisions." SAS Institute, 1 May 2019, https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3496-2019.pdf.

Lupi, Renato. "Pareto Example of Integration of SAS Visual Analytics with SAS Jobs via Data-Driven Content-Part 4." SAS Communities, SAS Institute, 22 Sept. 2020, https://communities.sas.com/t5/SAS-Communities-Library/Pareto-Example-of-Integration-of-SAS-Visual-Analytics-with-SAS/ta-p/681149.

SAS Institute. "Concepts: Open-Source Models." SAS® Model Manager 15.3: User's Guide, SAS Institute, 1 July 2020, https://go.documentation.sas.com/?docsetId=mdlmgrug&docsetTarget=n04i7s6bdu7ilgn1e350am3byuxx.htm&docsetVersion=15.3&locale=en.

Stern, David. "Examples of Row-Level Security in SAS Viya." SAS Support Communities, SAS Institute, 2 Aug. 2019, https://communities.sas.com/t5/SAS-Communities-Library/Examples-of-row-level-security-in-SAS-Viya/ta-p/578700?nobounce.

Wicklin, Rick. "The Missing Value Trick for Scoring a Regression Model." The DO Loop, SAS Institute, 17 Feb. 2014, https://blogs.sas.com/content/iml/2014/02/17/the-missing-value-trick-for-scoring-a-regression-model.html.

ACKNOWLEDGMENTS

Arne's Acknowledgements

I would like to thank the SAS communities for the valuable support they always dispense even if the question is very innocent.

The seed for everything I know in SAS is Rick Wicklin's Do Loop blog https://blogs.sas.com/content/iml/ . It is my primary source for learning SAS whilst building the foundation for the mathematical reasoning behind.

Stu's Acknowledgements

My wonderful wife Annie, for your love and support. I am proud to be your husband.

My manager Ned Maran, for your continued support and encouragement to write.

My mentor Jared Peterson, for helping me challenge myself and go further.

Johnny Stover, our local Job Execution Service guru. I don't think there's a problem we can't solve together with a 30-minute screen sharing session. Let's try world peace.

Vincent Delgobbo, the Job Execution Genius. Thank you for the detailed one-on-one help!

RECOMMENDED READING

SAS Job Execution Web Application 2.2: User's Guide

JSON Libname Engine Documentation

Reading data with the SAS JSON libname engine

Using Parameters with Data-Driven Content in SAS Visual Analytics by SAS's Renato Luppi

SAS Data Driven Content SDK Documentation

CONTACT INFORMATION

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

Arne Cordes
arne@campus.eae.es

https://www.linkedin.com/in/arnecordesbcn/

Stu Sztukowski
stu.sztukowski@sas.com

https://linkedin.com/in/StatsGuy

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.