Paper SAS4138-2020

Data Entry in SAS® Visual Analytics: Is It Possible?

Xavier Bizoux, SAS Institute Inc.

ABSTRACT

Achieving data entry within SAS® Visual Analytics can be a challenge for many customers. SAS Visual Analytics offers a way to achieve it using the data-driven content object and SAS® Viya® jobs. This session presents the available techniques that can be used to integrate a data-entry form into a SAS Visual Analytics report. It covers the basic principles of data-driven content objects and the different approaches to storing the data. At the end of the session, you should be able to create a simple data-entry form, insert it in a SAS Visual Analytics report, and store the data in SAS® Cloud Analytic Services (CAS) for later use.

INTRODUCTION

When building a data-entry application using SAS® Viya®, you should keep multiple dimensions in mind:

- User interface: how the user will interact with the application?
- Data storage: where the data will be stored?
- Integration: how will the application be integrated with SAS® Visual Analytics?

This paper will cover these three dimensions. The integration is done through the data-driven content object within SAS Visual Analytics. The storage is the SAS® Cloud Analytic Services (CAS) and the user interface is based on HTML and SAS Viya jobs.

If the terms above are new to you, don't worry, I will introduce each topic and build the application using a step-by-step approach.

Even though we will be using SAS® code, HTML, CSS, and JavaScript, you do not need to be a programming master to understand the concepts and to build your own application. Nevertheless, having a common understanding of the web development and SAS programming will surely help you in creating more advanced applications.

Before we dive into the different dimensions, let's review the different components that will be used throughout this paper. The following descriptions come from the SAS documentation.

SAS Visual Analytics provides a complete platform for analytics visualization, enabling you to identify patterns and relationships in data that weren't initially evident. Interactive, self-service BI and reporting capabilities are combined with out-of-the-box advanced analytics so that everyone can discover insights from any size and type of data, including text.

A data-driven content object enables you to display your data in a custom third-party visualization within your SAS Visual Analytics report. The third-party visualization can be authored in any JavaScript charting framework, such as D3.js, Google Charts, or CanvasJS to list a few examples. The visualization in a data-driven content object receives its data from SAS Visual Analytics and responds interactively with filters, ranks, and actions in the same way as other objects in your report. In addition to third-party visualizations, you can display any web content that is capable of being displayed in an IFrame.

A SAS Viya job consists of a program and its definition, which includes information such as the job name, author, and creation date and time. You can use jobs for web reporting, performing analytics, building web applications, and delivering content to clients. One such client is the SAS Job Execution Web Application. This client works with jobs that contain SAS code. These jobs can access any SAS data source or external file and create new tables, files, or other data targets that are supported by SAS. By default, jobs that are submitted through the SAS Job Execution Web Application begin with the %JESBEGIN macro and end with the %JESEND macro. These macros produce HTML output by default. An input form can be created from HTML or the SAS prompting interface to provide a user interface to the job. The job definitions and user interface definitions are assigned unique identifiers, stored in the SAS Infrastructure Data Server, and executed in real time by client applications.

SAS Cloud Analytic Services (CAS) is a server that provides the cloud-based, run-time environment for data management and analytics with SAS. Suitable for both on-premises and Cloud deployments, CAS uses a combination of hardware and software where data management and analytics take place on either a single machine or as a distributed server across multiple machines.

THE LOGIC

In this paper, I create a quiz application that displays questions with multiple choices and gives a hint about the answer(s) based on historical data. The quiz consists of three questions. The user navigates through the questions to answer them. At the end of the quiz, the user has the opportunity to review his/her answers before submitting.

WHAT IS THE BUSINESS PROCESS?

1. The user opens a VA report.
2. The report displays the first question with the different answer choices.
3. The user answers the first question by submitting a selected choice.
4. The next question is displayed.
5. The user answers the question by submitting a selected choice.
6. …
7. The user answers the last question by submitting a selected choice.
8. A summary of the answers is displayed.
9. The user reviews the answers and submits all choices.

WHAT IS HAPPENING ON THE BACKEND?

1. When the report opens, the data-driven content object retrieves data from the quiz history and calls the web page that will display the quiz question.
2. The web page calls a SAS Viya job, which copies the questions and possible answers from a public table to a private table for each user. The private table will be used to store the user's answers until the quiz is submitted.
3. The job returns the first question with the possible answers in the form of a JSON

object.

4. When submitting the first answer, the job receives the choice(s) and processes it to store the information in the private table.
5. The job returns the next question.
6. …
7. When submitting the last answer, the job processes the user choice.
8. The content of the user's private table is pushed to a public table, which stores the history of the quiz for all users.
9. The job returns the quiz summary to the client application.
10. The private table is then deleted.

As you can see, the process is quite simple. Now I will discuss the SAS Viya implementation pieces.

THE USER INTERFACE

SAS Viya jobs offer different techniques to pass parameters to the SAS code:

- URL parameters
- HTML form
- Job prompts

URL PARAMETERS

This is by far the simplest approach. You can call the SAS Job Execution application by specifying the program location from the SAS Content directory and specifying the parameters created in the SAS Viya job definition.

Here is a sample URL:

http://host:port/SASJobExecution/form/?_program=/Folder1/myJob&p1=123&p2=abc

URL	Role
host	Hostname of the SAS Viya environment.
port	The port number for the web server.
_program	The path to the program in SAS Content.
p1	A numeric parameter defined in the job.
p2	A character parameter defined in the job.

Table 1. URL Components for SAS Job Execution

For more information about executing jobs passing job definition parameters:

https://go.documentation.sas.com/?docsetId=jobexecug&docsetTarget=n1l0yij8rjbjqcn187q0tifuyab1.htm&docsetVersion=2.2&locale=en#n17h0afpcivgc7n1vzo03f686u91

HTML FORM

This option is a more advanced technique as you need to build an HTML form and bind it to the SAS Viya job. The benefit of this technique is that you are grouping the SAS code and the HTML form in a single job definition. The HTML form is displayed whenever the job is executed, which makes it an easy way to interact with jobs.

Here is a sample HTML code to input parameters:

```
<!DOCTYPE html>
<head>
<title>Simple HTML Example</title>
</head>
<body>
<h1>Simple HTML Example</h1>
<form action="/SASJobExecution/" target="_tab">
  <input type="hidden" name="_program" value="$PROGRAM$">
  <input type="hidden" name="_action"  value="execute">
<label>Specify a name for the greeting: </label>
<input type="text" name="myname" value="World" required>
<br/>
<input type="submit" value="Run code">
</form>
</body>
</html>
```

You should specifically pay attention to the <form> tag that points to the SAS Job Execution URL. You have also two <input> tags with a type of *hidden*. Those specify which program should be executed. Besides these, it is a simple HTML form element.

Here is the HTML prompt displayed during development in SAS® Studio:

Figure 1. HTML Prompt

For more information about executing jobs using a job form:

https://go.documentation.sas.com/?docsetId=jobexecug&docsetTarget=n1l0yij8rjbjqcn187q0tifuyab1.htm&docsetVersion=2.2&locale=en#n1hg6hagfuox76n1tnyitaxbv9ej

JOB PROMPTS

Another option is similar to the HTML form. It was introduced in SAS Viya 3.5 and SAS Job Execution 2.2. Instead of defining the prompts using HTML code, they are defined using XML.

Here is a sample XML code for the prompt:

```
<?xml version="1.0" encoding="UTF-16"?>
<Task schemaVersion="7.2">
<Registration>
<Name>Data Source Prompt</Name>
     <Description>Example of a Data Source Prompt</Description>
     <Version>5.2</Version>
</Registration>
<Metadata>
    <DataSources>
<DataSource name="DATASOURCE" where="true"> </DataSource>
    </DataSources>
<Options>
<Option inputType="string" name="DATATAB">DATA</Option>
<Option inputType="string" name="DATAGROUP">DATA</Option>
</Options>
</Metadata>
 <UI>
<Container option="DATATAB">
<Group open="true" option="DATAGROUP">
                  <DataItem data="DATASOURCE"/>
          </Group>
    </Container>
</UI>
</Task>
```

Here is the XML prompt displayed during development in SAS Studio:

Figure 2. XML Job Prompt

For more information about defining job prompts in SAS Studio 5.2:

https://go.documentation.sas.com/?docsetId=webeditorjobsdg&docsetTarget=n1eb1wa10cx5vsn1mfovntu8icqf.htm&docsetVersion=5.2&locale=en#p1qzdgfzoc3e2mn1kmqhdndastsu

MY CHOICE FOR THIS APPLICATION

I chose a hybrid version where I use an HTML form but don't add it to the job definition. The driving reason to maintain the HTML page on its own is that I prefer to build a single page web application. If I bind the HTML page to the SAS Viya job definition, each job execution will refresh the entire web page. Whereas updating only a subset of the web page provides for a more a seamless user experience.

DEVELOPING THE HTML CODE

The HTML code is short and can be downloaded from this GitHub (https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4138-2020-Bizoux/index.html).
The code is using the Bootstrap framework to manipulate the HTML page and to ease the cross-browser and cross-device development. For more information about Bootstrap, please refer to https://getbootstrap.com/
The most important part of the HTML is the form element:

```
<form id="quizForm" action="http://va85.gel.sas.com/SASJobExecution/"
      target="_SASResults">
        <input type="hidden" name="_program"
      value="/gelcontent/Jobs/SGF/DataEntry">
        <input type="hidden" name="_action" value="execute">
        <input type="hidden" name="_output_type" value="json">
        <input type="hidden" name="id" value="0">
        <details id="hintSection" class="row d-none">
            <summary class="btn btn-info">Hint!</summary>
            <br>
            <p>Other users answered :</p>
            <ul id="hintInfo"></ul>
        </details>
        <hr>
        <button id='submitBtn' type="submit" class="row btn btn-
      primary">Start</button>
  </form>
```

As you can see from the code, the form calls the SAS Job Execution web application when clicking the submit button. The form has some hidden parameters that are used to define:

Option	Role
_program	The location of the program in the SAS Content.
_action	The type of action that should be executed on the server side.
_output_type	The type of output that will be generated by the job. In this case a JSON formatted output.
id	The id of the question that is currently processed.

The form doesn't contain any element that will be displayed to the user. Those elements have to be added dynamically when the HTML receives the JSON output from the SAS Viya job. All that logic is handled through JavaScript code in the main.js file referenced at the bottom of the HTML page.

DEVELOPING THE JAVASCRIPT CODE

To be dynamic, the web page needs to include JavaScript. Even though we could use plain JavaScript, Bootstrap relies on jQuery. We will use jQuery to dynamically generate the content of our webpage. If you need more information about jQuery, please refer to https://jquery.com/

You can find the JavaScript file on GitHub (https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4138-2020-Bizoux/src/main.js).
As the focus of this paper is not explaining how to code in JavaScript, I will only cover the two functions required for this example from the main.js file.
In the submit function, the first section of the code manipulates the HTML page and defines what is displayed when the form is submitted. The second section handles the form submission. It reads the form fields and generate the HTTP request that calls the SAS Viya job. The *posting.done* function processes the response from the job and generates the HTML elements based on whether a question should be displayed or the summary.

I explain the second function in the Integration section of this paper.

THE DATA STORAGE

We've seen how the user interface has been created using HTML and JavaScript. The time comes now to cover the data storage and how to interact with the data on the back end. This is where the SAS Viya jobs are in use and SAS code is executed.

Before I cover the actual the SAS code, here are some concepts and options that you should know in order to make your decision while building your application. The main question is:

Where should I store my data?

The answer is not black or white, CAS or RDBMS.

The answer depends on the application you are building and its usage. If the application is only used by a few users and they need to add data or update data stored in a CAS table, I would recommend using a CAS table to store your data. Using CAS is a simple solution if you are trying to build a proof of concept type scenario and you don't have access to an RDBMS.

As soon as you need to build a production level application that requires concurrent user access, you should opt for an RDBMS. There are a few good reasons for that:

- Databases are designed for simultaneous user access.
- Databases are designed for Create, Read, Update, and Delete interactions (also known as CRUD).
- Databases have commit and rollback functionalities.

Knowing this, you may surmise that this paper demonstrates the use of an RDBMS, but this is not what I chose. The purpose of this paper is to demonstrate functionalities of SAS Viya. Therefore, I decided to use CAS storage. Even if this is not the perfect solution based on the considerations I outlined above, this is a quick solution to set up. It doesn't require configuration or access to an RDBMS and it can be used to demonstrate the functionalities of CAS.

If you want to know how to update and append data to a Global CAS table, I suggest reading the following article: https://blogs.sas.com/content/sgf/2017/11/15/concurrent-data-append-and-update-to-a-global-cas-table/

THE SAS CODE

The decision is to use the CAS server to store the data. We need SAS code to do the heavy lifting that was described in the section: What is happening on the backend? Here are the steps:

1. When the report opens, the data-driven content object retrieves data from the quiz history and calls the web page that will display the quiz question.
2. The web page calls a SAS Viya job, which copies the questions and possible answers from a public table to a private table for each user. The private table will be used to store the user's answers until the quiz is submitted.
3. The job returns the first question with the possible answers in the form of a JSON object.
4. When submitting the first answer, the job receives the choice(s) and processes it to store the information in the private table.
5. The job returns the next question.
6. …
7. When submitting the last answer, the job processes the user choice.
8. The content of the user's private table is pushed to a public table, which stores the history of the quiz for all users.
9. The job returns the quiz summary to the client application.
10. The private table is then deleted.

These steps will be written in SAS code in the form of SAS macros. The SAS code is written using SAS Studio and saved as a SAS Viya job.

The SAS code has four macros:

- Init: executed at the beginning of the job execution.
- Finish: executed after the last question is answered.
- Record: executed for each answer to record it.
- Output: generates the data in a JSON that will be used to display in the user interface.

You can find the SAS code on Github (https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4138-2020-Bizoux/DataEntry.sas). I will explain, in the following sections, each macro code.

Before going to the macro code in detail, we should make sure that the parameters passed from the user interface are defined. This is the role of the first line of the code:

```
%global id maxid answer01 answer02 answer03 answer04 answer05;
```

That line defines the required global macro variables for each variable that will be passed in the URL when the user interface calls the job. We also need to define other parameters as seen in Figure 3:

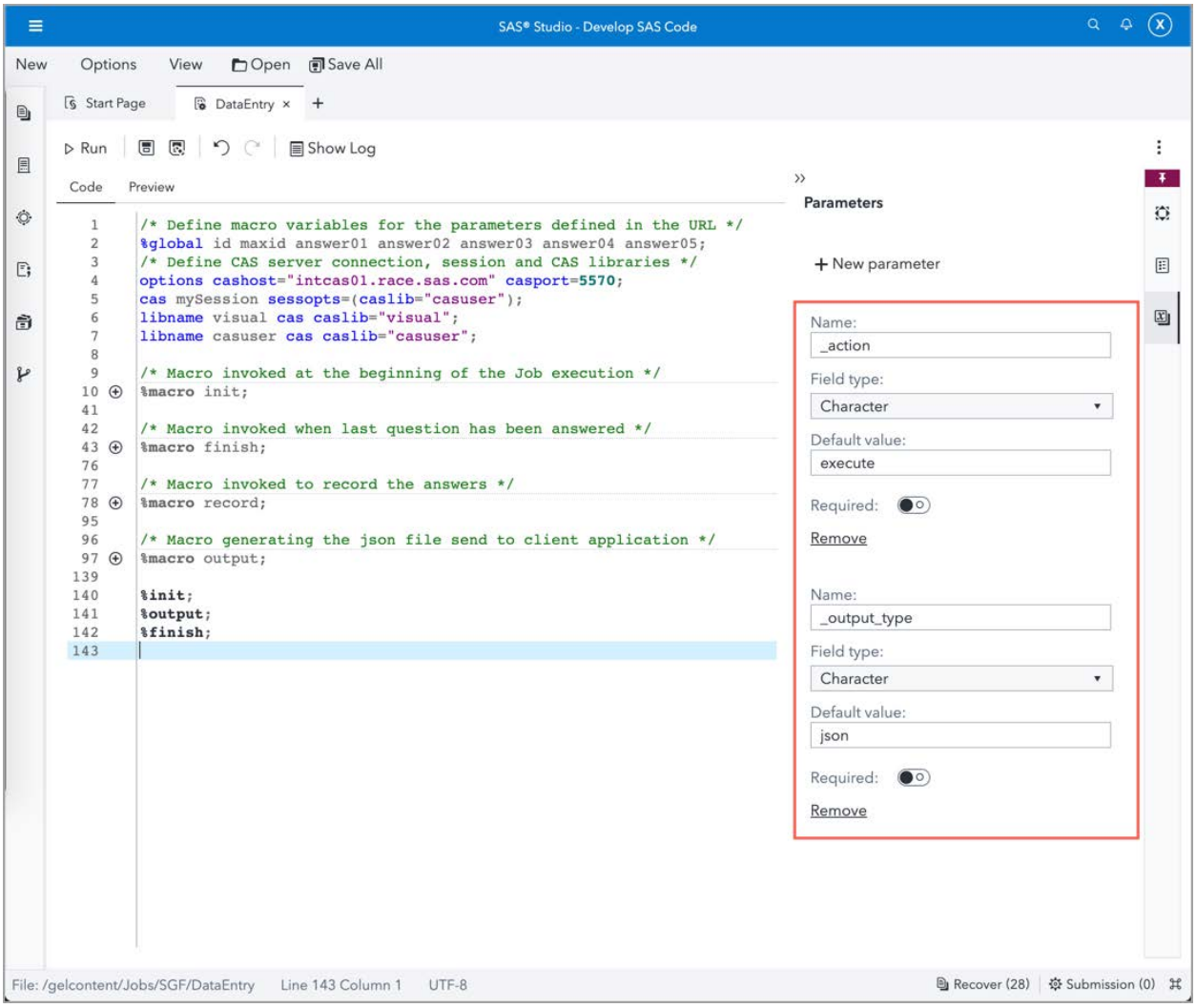

Figure 3. Defining Parameters in SAS Viya Jobs

Each macro is reading or writing to one or more CAS tables. Therefore, we need to define a connection to the CAS server and assign the different CAS libraries that we use.

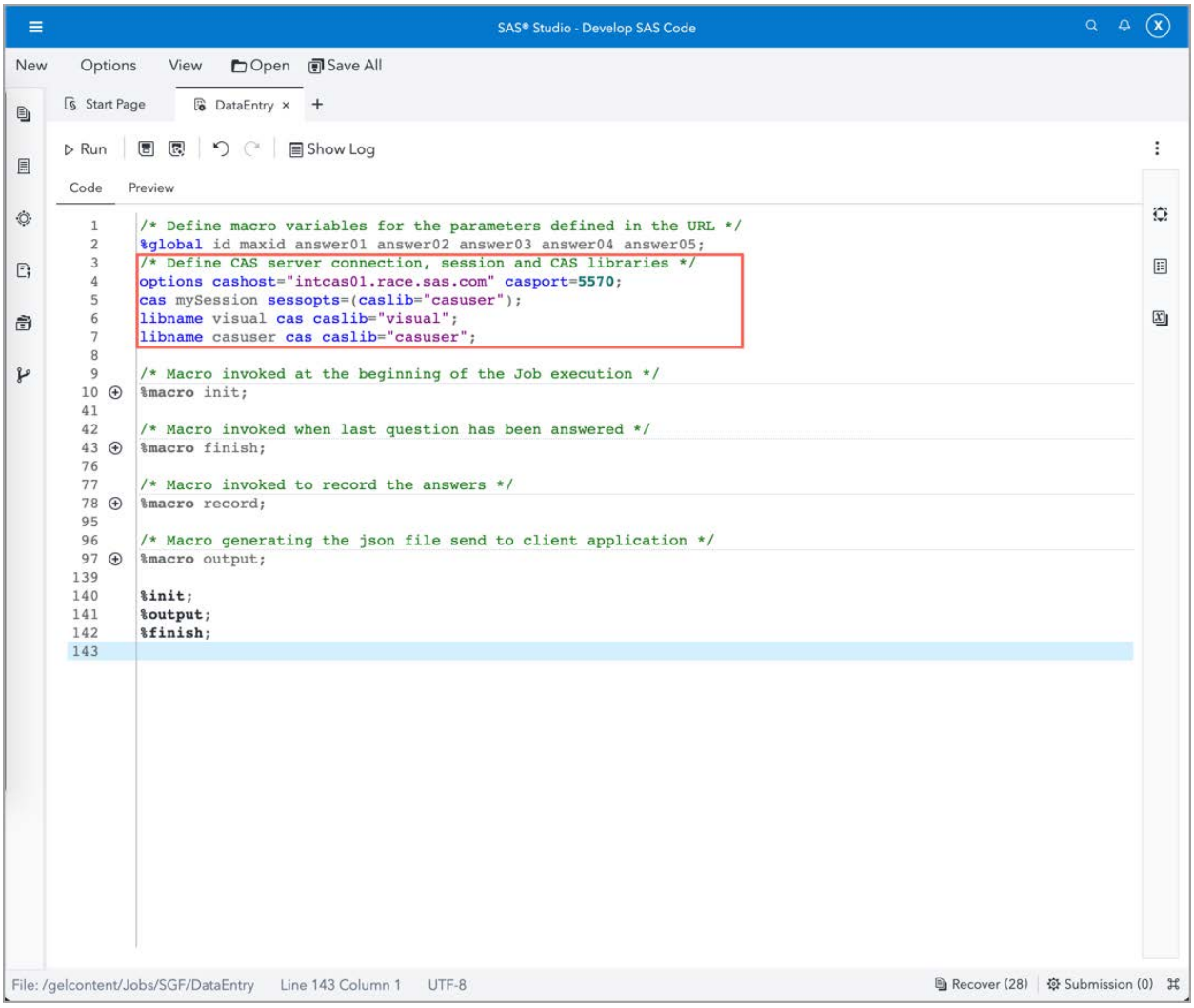

Figure 4. Define a Connection to the CAS Server and Assign the CAS Libraries

The code assigns two libraries that point to CAS libraries. The *Visual* CAS library contains:

- QUIZ_PROGRAMMING: the list of questions
- QUIZ_ANSWERS: the history of the answers.

The *Visual* CAS library is a global CAS library and the tables are also loaded with a global scope. It means that they are permanently loaded into CAS and are available to authorized users based on the security patterns defined in SAS Viya.

The CASUSER CAS library stores the data while the end user answers the quiz. It is an equivalent to the SASUSER library that is available in SAS Foundation. The tables loaded into that library are persisted between CAS sessions only if the table is promoted to the global scope. The benefit of the CASUSER CAS library is that the tables are only available to the user launching the CAS session and I referred to this earlier as the user's private table.

Now that you have a better understanding about the different CAS libraries and the tables that are used by the application. It's the right time to explain the macros and their purpose.

Init macro

```
/* Macro invoked at the beginning of the Job execution */
%macro init;
    %if %symexist(id) and %eval(&id > 1) %then
        %do;

            /* Call record macro to register the answers */
        %record;

            /* Increment question id */
        %let id = %eval(&id + 1 );
        %end;
    %else
        %do;

            /* Code executed when loading the first time */
        %let id = 1;

            data casuser.questions;
                set visual.quiz_programming;
            run;
            proc cas;
                table.promote /caslib="casuser" name="questions";
            quit;
        %end;

    /* Define the number of questions */
    proc sql noprint;
        create table maxid as select max(id) as maxid from casuser.questions;
        select maxid into :maxid from maxid;
    quit;

%mend;
```

Figure 5. The Init Macro

The INIT macro is executed at the beginning of each job execution. The role of this macro is to prepare the tables for processing. If this is a first execution, the ID variable is equal to 0 and the macro copies the list of questions from VISUAL.QUIZ_PROGRAMMING to the CASUSER library. The table is promoted to the global scope as this will make the table persistent between each job execution.

Subsequent executions will call the RECORD macro.

As a last step in the macro, the number of questions in the quiz is calculated. This value will be used to evaluate if the last question of the quiz has been reached.

Finish macro

```
/* Macro invoked when last question has been answered */
%macro finish;
    %if %eval(&id > &maxid) %then
        %do;
            /* Generate the data set to be loaded into the Global CASLib used for reporting */
            data casuser.toLoad;
                set casuser.questions (drop= questions choices hint justification);
                userid = "&sysuserid";
                datetime = datetime();
                format datetime datetime16.;
            run;
            /* Load data into the CASLib used for reporting and perform some cleanup activities */
            proc cas;
                table.tableExists result=res/caslib='visual' name='quiz_answers';

                if res.exists > 0 then
                do;
                    datastep.runcode result=dsResult /code="data visual.quiz_answers (append=yes); set casuser.toLoad; run;";
                end;
                else
                do;
                    table.promote / caslib='casuser' name='toLoad' target='quiz_answers' targetlib='visual';
                    table.promote / caslib='visual' name='quiz_answers';
                end;
                table.save / caslib='visual' name='quiz_answers' table={caslib='visual' name='quiz_answers'} replace=true;
                table.dropTable / caslib='casuser' name='questions';
                table.deleteSource / caslib='casuser' source='questions';
                table.dropTable / caslib='casuser' name='toLoad';
                table.deleteSource / caslib='casuser' source='toLoad';
            quit;
            /* Terminate the CAS session */
            cas mySession terminate;
        %end;
%mend;
```

Figure 6. The Finish Macro

The FINISH macro is executed when the id macro variable representing the question id is larger than the number of questions. When this is the case, the CASUSER.QUESTIONS table is enriched with the user ID and a date time stamp. When this is done, the table is appended to the VISUAL.QUIZ_ANSWERS table and some cleanup activities occur.

Record macro

```
/* Macro invoked to record the answers */
%macro record;
    proc cas;
        questionsTbl.name="questions";

        do i=1 to 5 by 1;
            name_i=put(i, z2.);
            varName='Answer'||name_i;

            if symget(varName) > "" then
                do;
                    questionsTbl.where="&id = id and name ='"|| varName ||"'";
                    table.update / table=questionsTbl set={{var='selected', value="1"}};
                end;
        end;
    quit;

%mend;
```

Figure 7. The Record Macro

The RECORD macro is executed when processing the answer from a user. The objective of the macro is to update the records in the CASUSER.QUESTIONS table based on the end-user answers. The RECORD macro is the only macro that is not called directly but as part of the INIT macro execution. All other macros are called independently.

Output macro

```
/* Macro generating the json file send to client application */
%macro output;
    %if %eval(&id > &maxid) %then
        %do;
            data questions;
                set casuser.questions;
            run;
            proc sort data=questions out=sortedQuestions;
                by id name;
            run;
            proc json out=_webout nosastags pretty;
                write open object;
                    write values "state";
                    write open object;
                        write values "numberOfQuestions" &maxid;
                        write values "summary" true;
                    write close;
                    write values "data";
                    write open array;
                        export sortedQuestions ;
                    write close;
                write close;
            run;
            quit;
        %end;
    %else
        %do;
            proc json out=_webout nosastags pretty;
                write open object;
                    write values "state";
                    write open object;
                        write values "numberOfQuestions" &maxid;
                        write values "summary" false;
                    write close;
                    write values "data";
                    write open array;
                        export casuser.questions (where=(ID=&id)) ;
                    write close;
                write close;
            run;
            quit;
        %end;
%mend;
```

Figure 8. The Output Macro

The OUTPUT macro is responsible for sending data in the form of JSON object to the client application (in this case our web page). In this example, the code is based on proc json and writes to the _WEBOUT destination.

THE INTEGRATION

At this stage of the proceeding, you should have a better understanding about the web application we are building and the underlying concepts. We have an HTML form that calls a SAS Viya job. The job returns some data that is then displayed in the HTML page that was calling the job.

The objective of this section will be to integrate the web application in SAS Visual Analytics using data-driven content object. The data from the report will be used to give the user a hint about the answers other users gave.

ADAPT THE REPORT DATA TO OUR NEEDS

The first stage will be to create a report based on the CAS tables stored in VISUAL CAS library. We will join the two tables QUIZ_ANSWERS and QUIZ_PROGRAMMING based on ID and NAME variables. At the same time, we select the variables that are available to our report.

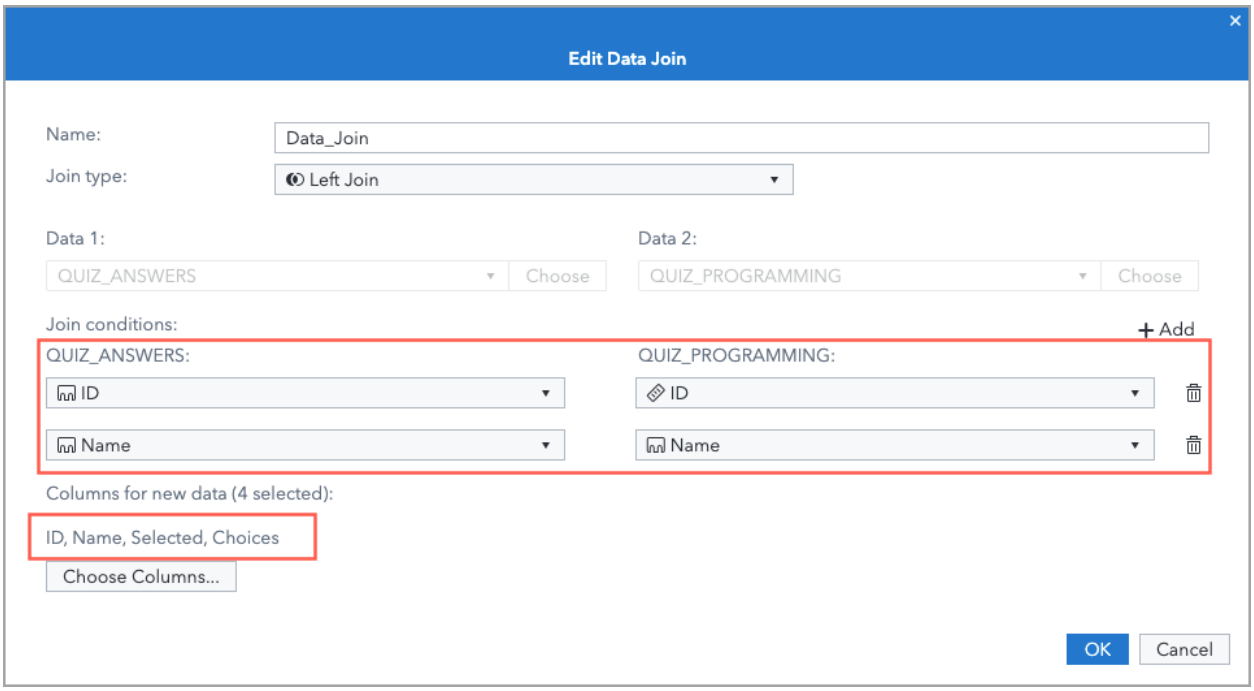

Figure 9. Create a Data Join in SAS Visual Analytics

We need also to change the ID variable classification from Measure to Category.

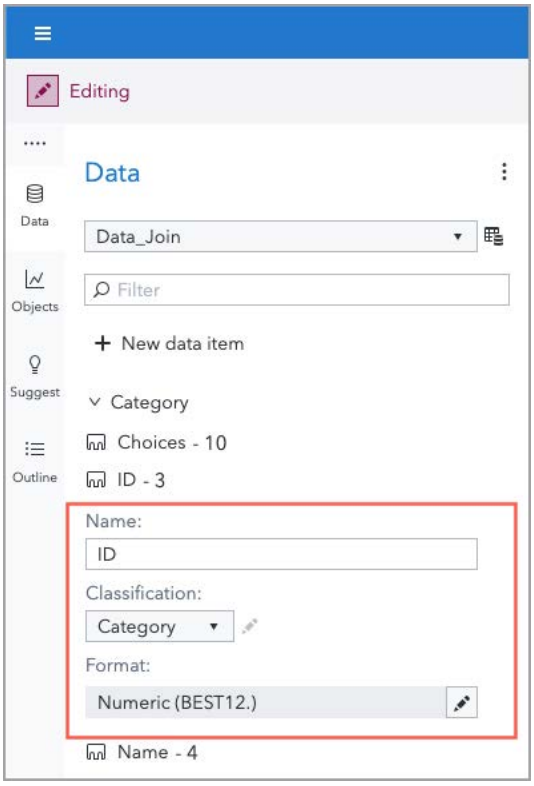

Figure 10. Change the Classification for the ID Variable

We should also create a new calculated measure that evaluates the selection ratio for a given choice in the quiz.

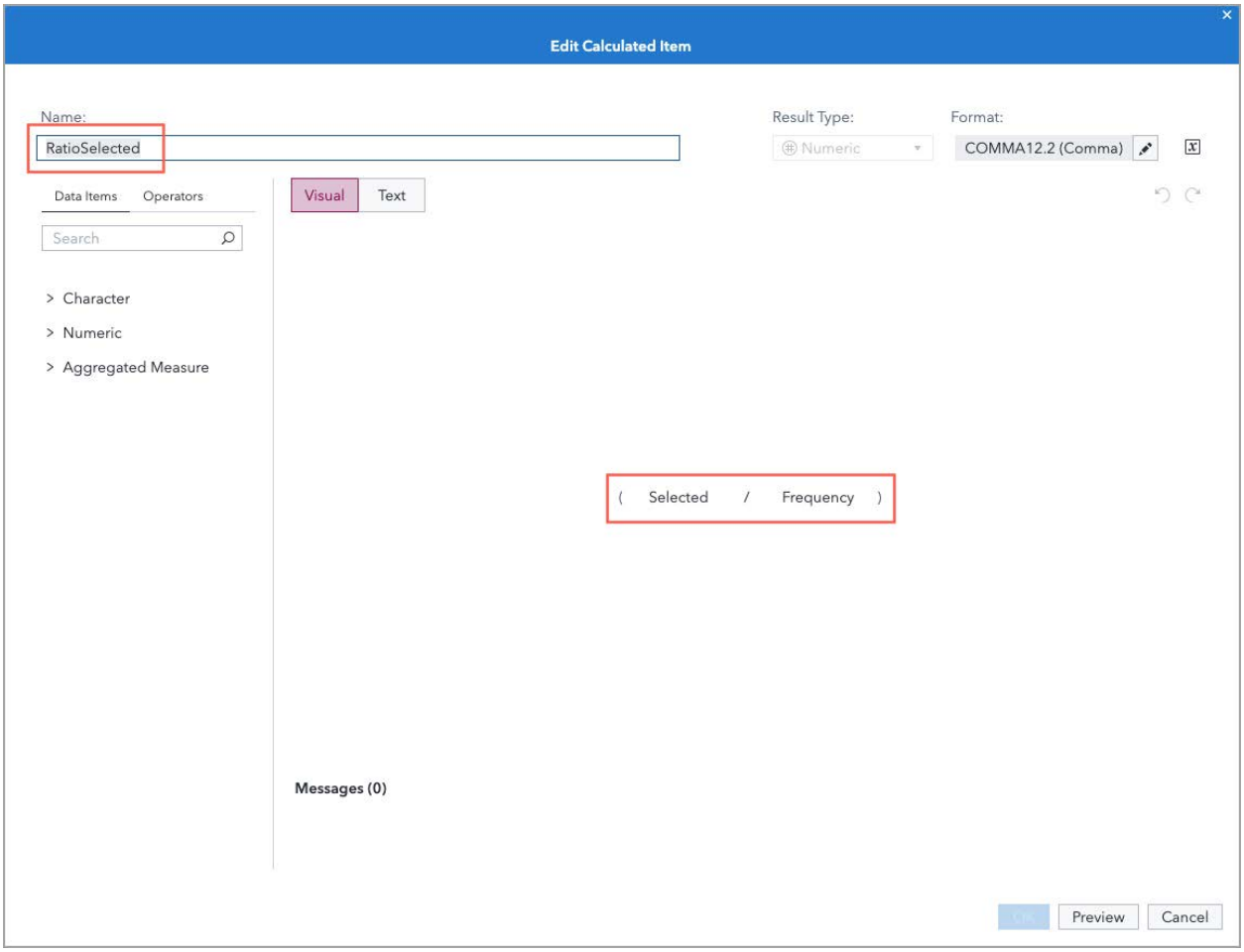

Figure 11. Create the RatioSelected Measure

By default, the calculated measure will have an aggregation type of Sum. We need to change that to Average to be relevant.

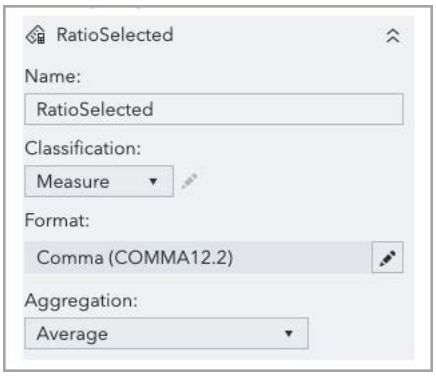

Figure 12. Change the Aggregation for the RatioSelected Measure

We can add the variables: ID, Choices, and RatioSelected to a list table in the report to check the result.

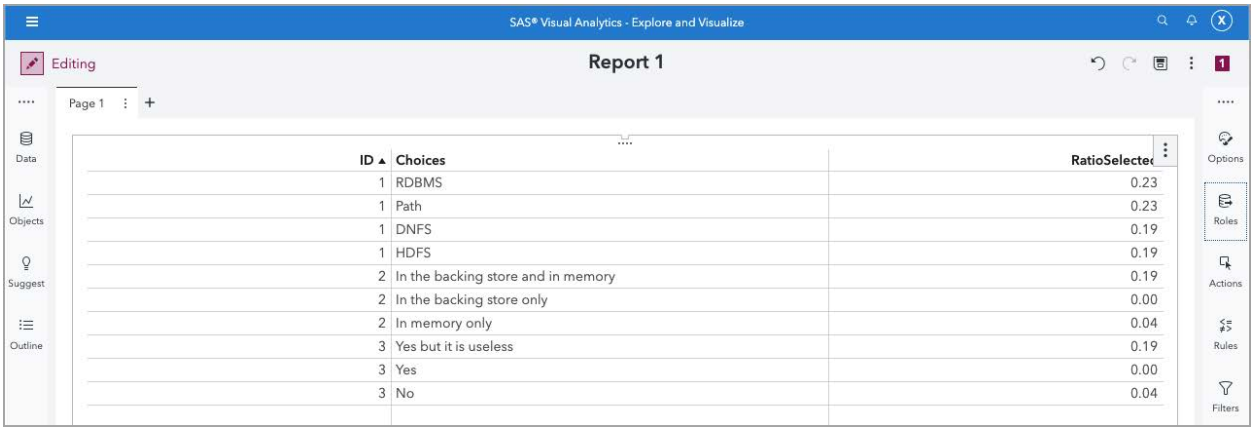

Figure 13. Check the Data in a List Table Object

We can now remove the list table object as we will not use it at a later stage. Instead of a list table object, we insert a data-driven content object. If you want more information about data-driven content objects and their usage, please refer to this article:

https://communities.sas.com/t5/SAS-Communities-Library/VA-Report-Example-Using-D3js-in-your-Report/ta-p/509033

We add the data-driven content object to the report canvas and we assign the ID, Choices, and RatioSelected variables to it. The report looks like this now:

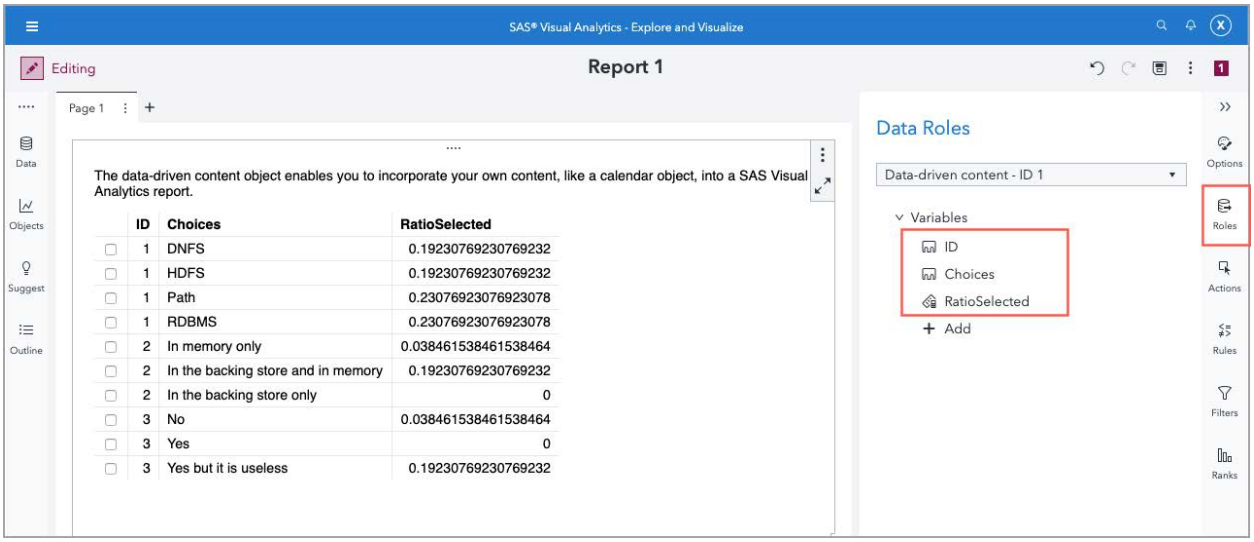

Figure 14. Report with Data-Driven Content Object

Please note that the variable order is important. I will explain why in the next section.

It is now time to link the data-driven content object to the web application we created. In order to do this, we first need to host the application on a web server. You can deploy the application on your own web server. In that case, you should make sure that the SAS Viya environment is configured properly to allow access from a different host. If you want more information about configuring your SAS Viya environment for cross-origin access, please refer to the following article:

https://communities.sas.com/t5/SAS-Communities-Library/Configure-Cross-Origin-Resource-Sharing-for-SAS-Viya-for-REST/ta-p/589015

To ease the deployment process and to reduce the configuration, you can also choose to host the web application on the web server that is installed and configured with SAS Viya. By default, only the root user can write under /var/www/html. You should request your system administrator to set the correct security in order to copy the web application under that location. Let's imagine you convinced the administrator and your application is deployed on the web server. The next step is to configure the data-driven content object to point to your web application instead of the default web application. We open the Options pane for the data-driven content object and we define the correct URL at the bottom:

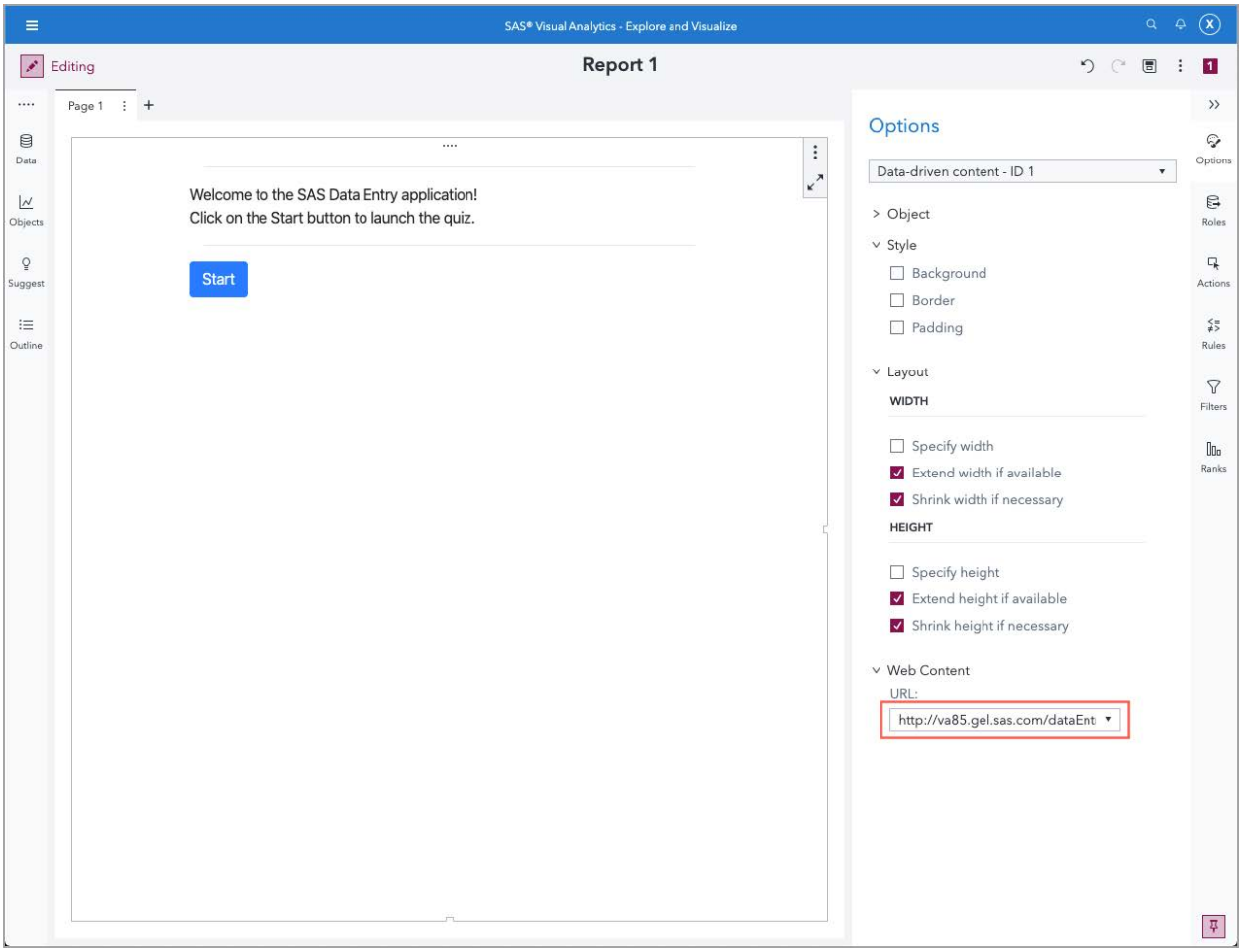

Figure 15. Define the Data-Driven Content URL

In this example, the URL is: http://va85.gel.sas.com/dataEntry. dataEntry is the directory where I copied the content of my web application under /var/www/html on the machine hosting the web server.

We can now start to use the web application and pressing the Hint! Button will display the percentages each answer received from previous users. If you copied the code provided on Git (https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4138-2020-Bizoux), it should work. In the next section, I will explain how you can integrate data from a SAS Visual Analytics report in your web application.

INTEGRATE REPORT DATA INTO YOUR WEB APPLICATION

When you work with data-driven content object, data from the report are passed to the object in the form of JSON object. That JSON object can be easily manipulated using JavaScript. SAS provides prebuilt functions on GitHub that can be used to ease the usage of data-driven content and do the wiring for you.

The functions are available in form of two JavaScript files:

https://github.com/sassoftware/sas-visualanalytics-thirdpartyvisualizations/blob/master/util/contentUtil.js

https://github.com/sassoftware/sas-visualanalytics-thirdpartyvisualizations/blob/master/util/messagingUtil.js

Those files should be referenced in your index.html. You can therefore download the files and store them under the same Libs folder as the Bootstrap and jQuery libraries we used in the application.

```
<script src="libs/util/contentUtil.js"></script>
<script src="libs/util/messagingUtil.js"></script>
```

To handle data sent by the data-driven content object, you have a few lines of code in the main.js file.

```
var ddcData = null;

va.messagingUtil.setOnDataReceivedCallback(onDataReceived);

function onDataReceived (resultData){
        ddcData = resultData.data;
    }
```

The lines above will handle the data received from the data-driven content object.

The following lines are responsible for displaying the data in the hintSection div of the web application.

```
$('[name*="id"]').change(function (event) {
        var id = $('[name*="id"]').val();
        $("#hintInfo").empty();
        $("#hintSection").removeAttr("open");
        $.each(ddcData, function (key, val){
            if (val[0] == id) {
                var percent = Number(val[2]).toLocaleString(undefined,{style:
'percent', minimumFractionDigits:2});
                $("<li />", {"text": val[1] + " : " + percent}
).appendTo("#hintInfo");
            }
        })
    });
```

The final result is this:

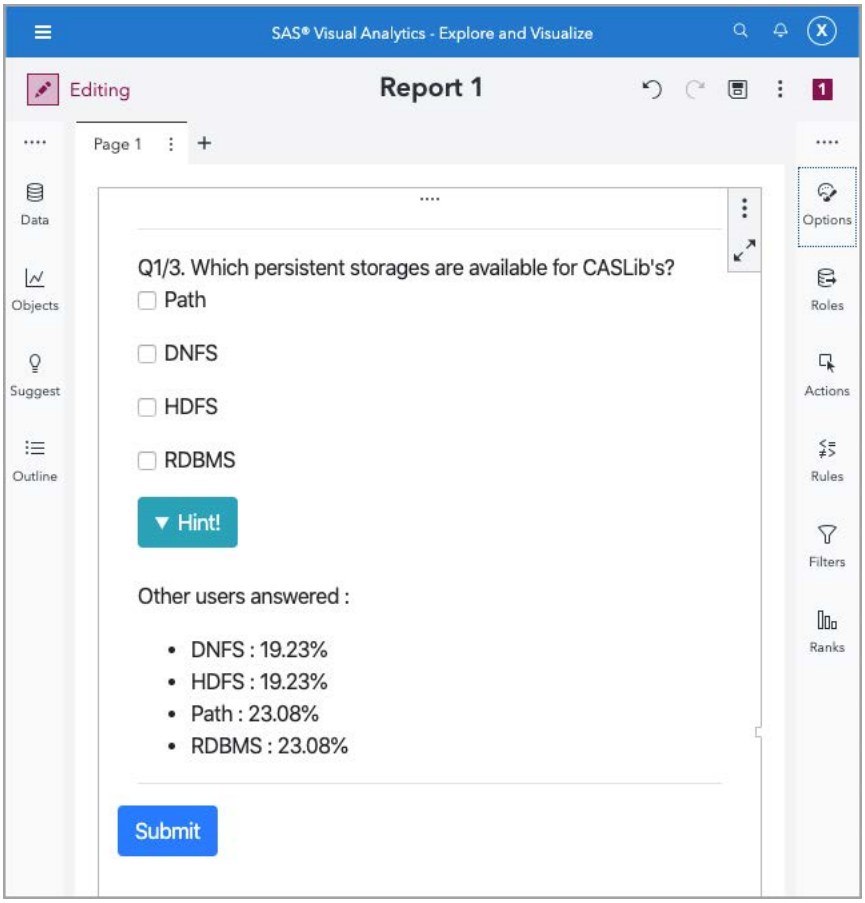

Figure 16. The Final Report

CONCLUSION

Using a combination of HTML, JavaScript and SAS code, you can build web applications that integrate with SAS Visual Analytics. The applications can perform various tasks, from simple visualization to data entry, and integrate results calculated by the CAS server. What you should keep in mind is that CAS is designed for calculation on large data tables. The main purpose is not to handle transactional data (though it can do it).

What is your next step? Test this for yourself and build a web application that suits your needs!

CONTACT INFORMATION

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

Xavier Bizoux
SAS Institute Inc. - Belgium
xavier.bizoux@sas.com