**PhUSE 2017**

**Paper AD01**

# A User Interface To Your SAS Programs

Hailemichael M. Worku, OCS Consulting, ‘s-Hertogenbosch, The Netherlands

## ABSTRACT

In our industry, SAS programs are usually plain text SAS scripts that are run in a manual or automated fashion. Neither way provides a convenient solution to provide input parameters, let alone any form of interactivity. A relatively accessible solution may be found in SAS Enterprise Guide Add-Ins. Add-Ins provide a familiar, customizable user interface that connects to custom SAS programs in the background. With an Add-In the user can interact with the programs and the data without having to have any affinity with programming at all. The user interface can display raw or processed data and results from reporting or statistical procedures, it can also produce permanent output such as rtf or pdf files. This paper will provide insight into the development of an Add-In using application examples.

## INTRODUCTION

SAS software is used in a wide range of sectors (e.g., pharmaceuticals, finance, R&D, etc) for data management, analysis and for other purposes. As any scripting language, which could have a steep learning curve, SAS software does also require some technical background from users. However, one can leverage the high analytical capability of SAS by using a Graphical User Interface (GUI) for the interaction and SAS for the backend. By doing this, non-technical users and data analysts who have little or no affinity to programming or any of the programming languages for that matter can use SAS for their needs.

OCS Consulting recently developed an application for one of its clients for analyzing motor activity data in animals. The application comprises a user interface built in Microsoft .NET Technology with a logical and statistical backend built in SAS. This paper will provide insight into the development of an Add-In using two examples: a hypothetical application built for converting user temperature values, and another one which is more similar to a real-world application for reading and analyzing data using machine learning techniques.

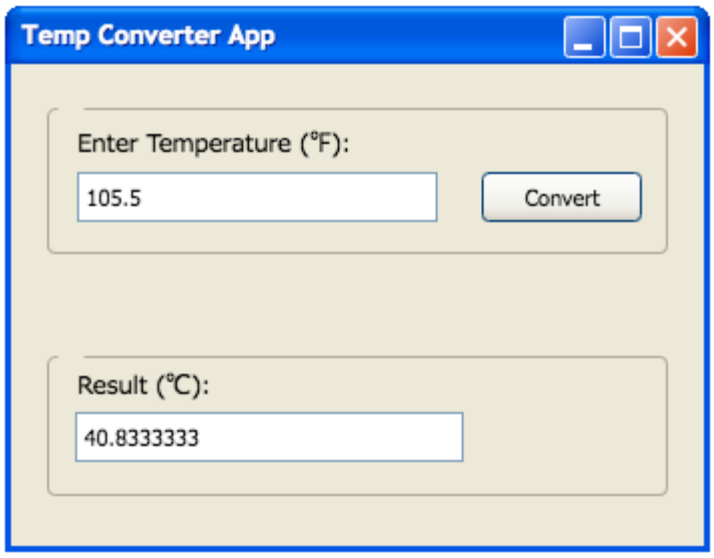
## TALKING TO SAS VIA GRAPHICAL USER INTERFACE

A GUI is a type of User Interface (UI) that allows users to interact with an application using elements such as buttons, checkboxes, radio-buttons, combo-boxes, etc, for performing certain tasks. In data management and statistical analysis for example, the user could; import/load raw data using a button, select/remove record(s) using a data grid, preprocess, explore, analyze data and finally generate reports using sliders, spinners, checkboxes, combo boxes, tab, etc.

An example of a GUI is shown below in the left panel of Image 1. The 'Temp Converter App' (TCApp) is a hypothetical example of an application which converts a user temperature input value from degree Fahrenheit (°F) to degree Celsius (°C). The TCApp interface has three main elements: (1) a text-field which allows user to enter input temperature value in °F; (2) a button which requests the conversion to start;(3) a text-field for displaying output results in °C once the computation is completed.

The right panel of Image 1 shows SAS programs that are run by the interface at the backend. When the user clicks the 'Convert' button, the interface communicates with SAS Workspace Server (SWS) and sends the programs together with user input values for execution. The SWS is responsible for interpretation, debugging and execution of SAS programs, in short it is the 'brain of the application'. The SAS Enterprise Guide (EG) software, which is another product of SAS, for example uses SWS at the backend for the same purpose. Once the execution is completed, the SWS sends information back to the interface such as output dataset, summary statistics, macro variables, file location of a graph or a report, etc. The interface then displays the result to the user.

As shown in Image 1, for example, the user entered a temperature value of 105.5 in the text-field 'Enter Temperature (°F):'. When the user clicked the button 'Convert', the TCApp application returned a value of 40.8333333 automatically and displayed it in the text-field 'Result (°C):'.




```
%let txtFld_tempInput = 105.5;

%include "temp_converter.sas";
```

*Image 1*: The left panel shows a mock-up of the Graphical User Interface (GUI) for Temp Converter App, and the right panel for SAS programs that is run at the backend.

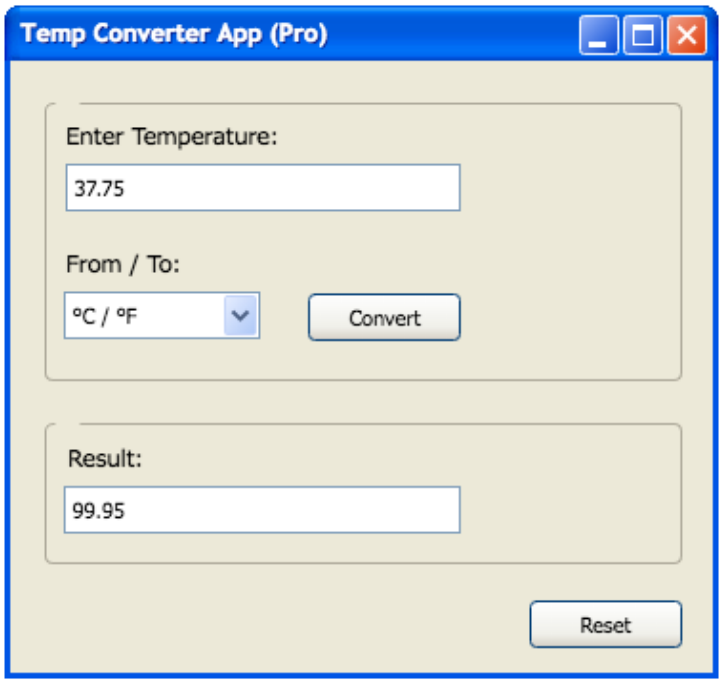
High-level programming languages such as Microsoft .NET Technology, C#, or Java, to mention but a few, are often used to develop GUIs. However, it doesn't matter for the backend (SAS) in which language the front-end/interface is built with. Moreover, without many changes for the backend the same SAS programs can be shipped to Web technologies to deliver the same functionality as an online application.

## EXTENDING THE TCAPP GUI: HOW MUCH WOULD IT COST FOR THE BACKEND?

The TCApp application shown in Image 1 is limited to a one-way conversion, i.e., only from °F to °C. It would be interesting if the TCApp application could also allow users to convert temperature not only from °F to °C, but also from °C to °F. As shown in the left panel of Image 2, a new combo-box and a reset button are added for this purpose. The combo-box allows the user to select the type of conversion to perform and it has two options: (1) °F / °C which is a default setting of the interface; and, (2) °C / °F. The interface sends a value of '1' or '2' to the backend (these values represent the two options of the combo-box) when the 'Convert' button is clicked.

The 'Reset' button is used to reset the whole interface's input values to their default settings (e.g., the input text-field is reset to null/blank, the combo-box is reset to °F / °C and the result text-field is reset to null/blank). The request of the 'Reset' button, however, is independent of SAS Workspace Server and is handled by the programing language used for building the interface itself (e.g., C# or Microsoft .NET, etc).

The right panel of Image 2 shows SAS programs (very like the one in Image 1) which are run by the interface at the backend. The main SAS program which is responsible for performing conversion of user input temperature is 'temp_converter_pro.sas' which is an extension of the same program presented in Image 1. The full SAS program of the TCApp (Pro version) is also displayed in Image 3. The program shows how easy it is to maintain and extend the backend for new functionality. The backend SAS program of TCApp shown in Image 3, for example, is capable of handling two versions of the application (trial version and pro version). Depending on the value of input parameter 'intrfcToSAS_runPro' which is sent by the interface to the backend, the SWS calls and executes either the trial version or pro version of the SAS program. When it is set to 'y', the Pro version is run; otherwise, the Trial version would run at the backend. Note that the macro variable 'interfaceToSAS_runPro' is not set by the user, it is rather a value sent by the interface itself based on the permission level of the user.




```
%let txtFld_tempInput = 37.75;
%let comboBox_fromTo = 2;

%include "temp_converter_pro.sas";
```

*Image 2*: The left panel shows a mock-up of the Graphical User Interface (GUI) of Temp Converter App, the Pro version. The right panel shows SAS programs that are run at the backend.

```
/*******************************************************************************
  Environment setting.
*******************************************************************************/
%global dir_app;

%let dir_app = ...\PhUSE\Temp Convertor App;
filename prg_app "&dir_app.\programs";

/*******************************************************************************
  Get user inputs sent by an interface (e.g., dotNET, C# or Java, etc).
*******************************************************************************/
%let intrfcToSAS_runPro = y;
%let intrfcToSAS_txtFld_tempInput = 37.75;
%let intrfcToSAS_comboBox_fromTo = 2;

/*******************************************************************************
  The Temp Convertor App (TCApp).
*******************************************************************************/
%macro TCApp(app_runPro =,
             debugme    =);

    %if       %upcase(&debugme.) eq Y     %then %do;
      options mprint mlogic symbolgen;
    %end;
    %else %if %upcase(&debugme.) eq N or
              %upcase(&debugme.) eq       %then %do;
      options nomprint nomlogic nosymbolgen;
    %end;

    /* TCApp: The Trial version. */
    %if (%upcase(%sysfunc(strip(&app_runPro.)))) = N %then %do;

        /* Load and Run SAS convertor program (trial version). */
        %include prg_app(temp_convertor.sas);

        %put >>> ---------------------------------------- ;
           %put >>> TCApp (trial version) is running...  ;
           %put >>> ---------------------------------- ;
        %temp_convertor(temp_input = &&intrfcToSAS_txtFld_tempInput.);
    %end;

    /* TCApp: The Pro version. */
    %else %if (%upcase(%sysfunc(strip(&app_runPro.)))) = Y %then %do;

        /* Load and Run SAS convertor program (Pro version). */
        %include prg_app(temp_convertor_pro.sas);

        %put >>> ---------------------------------------- ;
           %put >>> TCApp (pro version) is running...    ;
           %put >>> ---------------------------------- ;
        %temp_convertor_pro(temp_input = &&intrfcToSAS_txtFld_tempInput.,
                            temp_fromTo = &&intrfcToSAS_comboBox_fromTo.);
    %end;
%mend TCApp;

/* Run TCApp program. */
%TCApp(app_runPro = &intrfcToSAS_runPro., debugme =n);
```

*Image 3*: The backend SAS programs that are ran by the TCApp application (Pro version).

## APPLICATION: THE TITANIC EXPLORER APP

The temperature conversion application program that was discussed before is easy to explain and understand. In real world applications however, more and different types of elements might be populated on an interface (e.g., radio-buttons, checkboxes, datagrids, sliders and spinners, to mention but a few) for performing different tasks. The mock-up below in Image 4 shows an example of a real world-like application. The Titanic Explorer App interface consists of three main compartments: (1) the 'Import Data' area where raw dataset is imported, (2) the 'Machine Learning (ML)' area where statistical analysis is performed on the imported dataset, and (3) the 'Output Window' area where output results are displayed for a user. The 'Machine Learning (ML)' area is subdivided into four main parts which will be discussed in depth shortly.

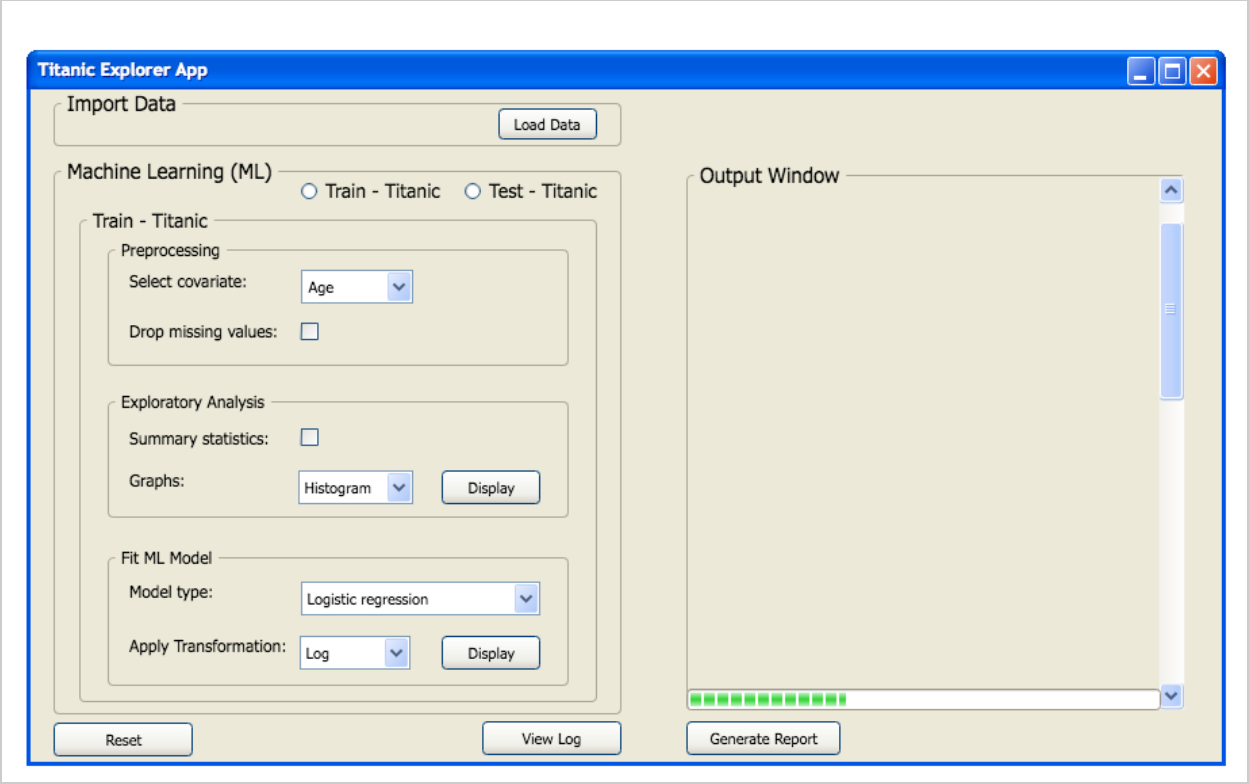



*Image 4*: A mock-up of the GUI for the Titanic Explorer App.

### TITANIC: MACHINE LEARNING FROM DISASTER

As part of a real-world application example, a famous dataset about survival on the Titanic was used, this is available online for free from the Kaggle website: https://www.kaggle.com/. Kaggle is a data science platform where companies contribute their data sets for free and data scientists and engineers from all corners of the world participate in a competition to come-up with the best predictive machine learning model(s) for solving companies' big problems (e.g., health, business, transportation, etc). The screenshot of the website is displayed below in Image 5, showing the survival on the Titanic competition. For this competition, about 7,763 data scientists have been contributing up till this point, the competition will continue for another 3 years. It is encouraged to be part of this developer group, learn about ML techniques and contribute too.

For survival on the Titanic competition, Kaggle provides two sets of data sets: the Training set and the Test set. These data sets have records on 2,224 travelers and crew, among who 1,502 were killed during the accident. In the Machine Learning field, the training data set is used to train different machine learning techniques such as logistic regression, tree regression, and random forest, to mention but a few. The Test data set is used to test the performance of the trained ML model on its predictive power and other metrics.

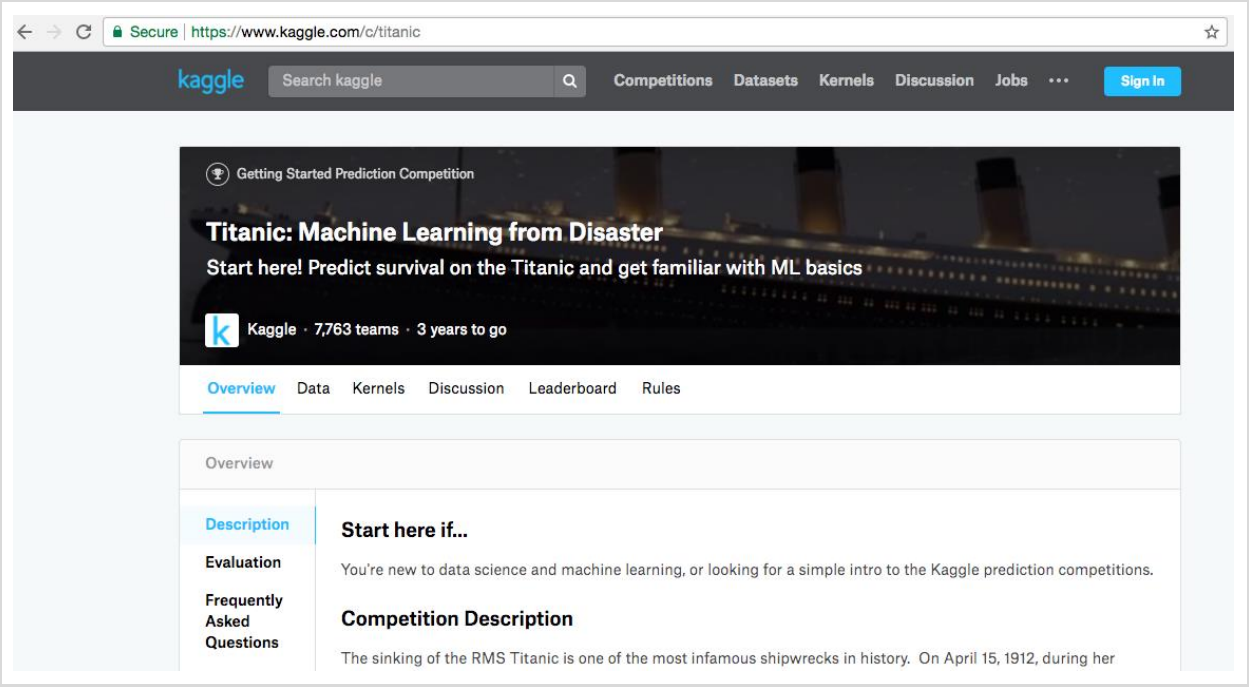



*Image 5*: Kaggle competition on predicting survival on the Titanic using Machine Learning (ML) techniques.

Data scientists often apply the following three steps to datasets for building/training ML models**:**

1. Preprocessing step: this is where most of data cleaning is done (e.g., excluding missing values and/or outliers, extracting new variables for analysis, etc).
2. Exploratory analysis step: this is where the data scientist tries to explore the data by looking for special trends and relationship with response variable (e.g., survived). Graphs (e.g., scatter plot, line plot, etc) and descriptive statistics are mainly used here.
3. Fitting ML models: this is the final step of the training process where a statistical model is fit to the processed dataset. After determining the final structure of the trained ML model, it will be stored for the next step (the testing phase). That is, the ML model will be applied on the Test data set to evaluate its predictive power.

The above three steps are also part of the mock-up of GUI for the Titanic Explorer App shown above in Image 4, as presented under the 'Train - Titanic' tab.

**THE PROCESS FLOW DIAGRAM**

The process flow diagram of the Titanic Explorer App is shown below in Image 6. The two data sets from Kaggle (Train.csv and Test.csv) are first read-in once by the application using the 'Load Data' button. After some processing, two SAS datasets (train and test) are created in the WORK library by the SWS. When the user clicks 'Train - Titanic' radio-button, the application populates the 'Train - Titanic' tab for the user to proceed with the three training steps that were discussed above (i.e., preprocessing, exploratory analysis and fitting ML models). Using the two 'Display' buttons, the user explores the data and assess the adequacy of the ML model by looking at the result populated in the 'Output Window'.

Once the user is satisfied with the trained ML model, the user would proceed to the testing phase where the trained ML model would be tested on the imported Test data set. When the user selects the 'Test - Titanic' radio-button, a new tab called 'Test - Titanic' would be populated on the interface. This would have its own set of elements for performing different tasks such as which validation techniques would be used for assessing the predictive power of the ML model (e.g., cross-validation, K-fold, etc). Before generating reports, the user might want to assess the execution process of each step, this is available in SAS log. The 'View Log' button of the interface can be used for this purpose and the log file is populated in the 'Output Window' when the 'View Log' button is clicked.

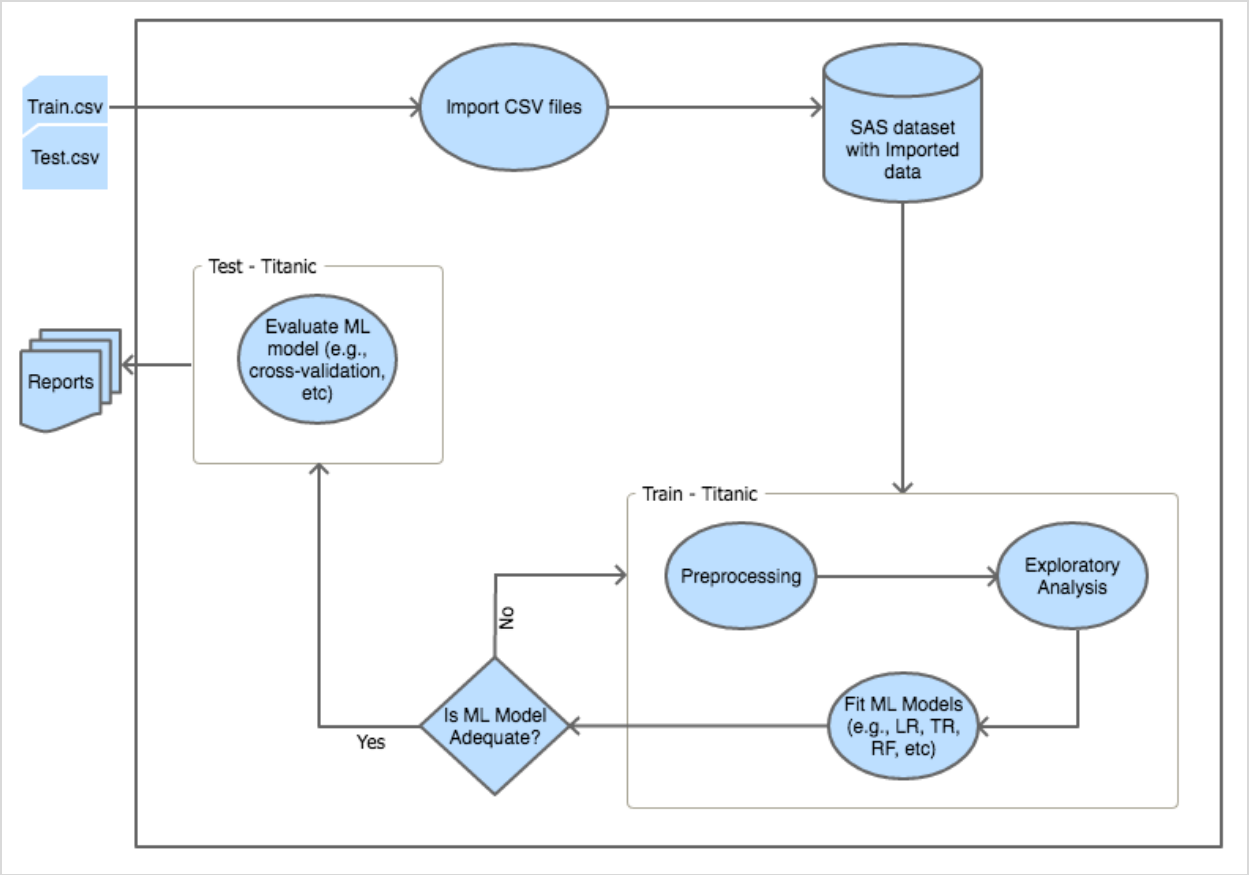



*Image 6*: The process flow diagram of the Titanic Explorer App.

Once the user is satisfied with the predictive power of the final ML model and with the log, the user might want to generate reports for a review/submission or to share with collaborators or friends. This can be done using the 'Generate Report' button of the interface. When the 'Generate Report' button is clicked, the Titanic Explorer App directs the user to a new Window which populates all available outputs so far, such as; summary statistics, graphs, estimates of ML model, residuals for further model assessment, etc. The user would then select some or all results that they would like to be present in the final report. This output Window might also include a 'Send Email' button for sharing the work with collaborators for a review or with friends.

**THE BACKEND USING SAS: TECHNICAL PART**

As shown in the process flow diagram above, different decisions and computations are performed at the backend of the application. In this example, SAS was used for the backend of the Titanic Explorer App to perform these tasks. The SAS programs that were used for prototyping the application are presented below in Image 7. Due to its modular nature, the backend is easy to maintain and upgrade with additional functionality. A modular design approach is a development approach where a system is subdivided into small parts by purpose. This example used a set of SAS macros at the backend of the Titanic Explorer App which are grouped as follows: (1) The ‘load_data’ macro which is responsible for importing the raw Comma Separated Value (CSV) files; (2) The ‘preprocess_data’ macro for preprocessing the raw dataset, (3) The ‘explore_data’ macro for exploring the processed dataset; (4) The ‘fit_MLModel’ macro for applying the ML model to the processed dataset. As shown in the process flow diagram above in Image 6, these macros are part of the training process.

The ‘TitExpApp’ is the main function of the application which determines whether the ML model is trained or tested based on what values are set for an input parameter 'intrfcToSAS_MLArea_RadioBtn', whose value is passed by an interface. If this parameter is set to a value of '1', the SWS calls and runs the training program, otherwise if it is set to '2', then the trained ML model is tested on the imported test dataset. Finally, the 'generate_report' program is run when the user clicks the 'Generate Report' button.

```
/*************************************************************************
   Environment setting.
*************************************************************************/
%global dir_app dir_rawdata dir_output;
%let dir_app = ...\PhUSE\Titanic Explorer App;
%let dir_rawdata = &dir_app.\rawdata;
%let dir_output = &dir_app.\output;

filename prg_app "&dir_app.\programs";

/*************************************************************************
   Import supportive macros.
*************************************************************************/
%include prg_app(load_data.sas);
%include prg_app(preprocess_data.sas);
%include prg_app(explore_data.sas);
%include prg_app(fit_MLModel.sas);
%include prg_app(test_MLModel.sas);
%include prg_app(generate_report.sas);


/*************************************************************************
   Get input values sent by an interface (e.g., dotNET, C#, or Java, etc).
*************************************************************************/
/* Application area 0: Load source dataset. */
%let intrfcToSAS_LoadArea_btnLoad = train.csv | test.csv;
%let intrfcToSAS_MLArea_RadioBtn = 1;

/* Application area 1: Preprocess imported dataset. */
%let intrfcToSAS_preprArea_comboBox = age;
%let intrfcToSAS_preprArea_chkBox = checked;

/* Application area 2: Data Exploration. */
%let intrfcToSAS_explArea_chkBox = checked;
%let intrfcToSAS_explArea_comboBox = histogram;

/* Application area 3: Apply Machine Learning (ML). */
%let intrfcToSAS_fitMLArea_chkBox = logistic;
%let intrfcToSAS_fitMLArea_comboBox = log;

/* Application area 4: Generate Report. */
%let intrfcToSAS_genRepArea_btnGR = yes;


/*************************************************************************
   Step 0. Load Titanic raw dataset.
*************************************************************************/
%load_data(raw_dsin = &intrfcToSAS_LoadArea_btnLoad.);


/*************************************************************************
   Titanic Explorer App.
*************************************************************************/
%macro TitExpApp(MLArea = );

    /* Training phase. */
    %if %sysfunc(strip(&MLArea.)) = 1 %then %do;
        /*****************************************************************
           Step 1. Preprocess raw Titanic data.
        *****************************************************************/
```

```
        %preprocess_data(dsin         = work.train,
                         varin        = &&intrfcToSAS_preprArea_comboBox.,
                         drop_missing = &&intrfcToSAS_preprArea_chkBox.,
                         debugme      = n);

        /*****************************************************************************
           Step 2. Explore processed dataset.
        *****************************************************************************/
        %explore_data(show_descStat = &&intrfcToSAS_explArea_chkBox.,
                      graph_type    = &&intrfcToSAS_explArea_comboBox.,
                      debugme       = n);

        /*****************************************************************************
           Step 3. Fit ML model on processed data.
        *****************************************************************************/
        %fit_MLModel(model_type           = &&intrfcToSAS_fitMLArea_comboBox.,
                     apply_transformation = &&intrfcToSAS_fitMLArea_comboBox.,
                     debugme              = n);
    %end;
    %else %if %sysfunc(strip(&MLArea.)) = 2 %then %do;
    /* Testing phase. */
/*        %test_MLModel(...);*/
    %end;
%mend TitExpApp;

/* Run the app. */
%TitExpApp(MLArea = &intrfcToSAS_MLArea_RadioBtn.);

/*****************************************************************************
   Step 4. Generate report.
*****************************************************************************/
/*%generate_report(...);*/
```

*Image 7*: The backend SAS programs that are run by the Titanic Explorer App.

## FROM DEVELOPMENT TO PRODUCTION: THE SAS ENTERPRISE GUIDE MANAGER

Once an application is fully tested and developed (both the interface and the backend), it goes to production. The production of your application and running of the final application is done using the SAS Enterprise Guide (EG). The SAS EG is an IDE dedicated for SAS development. The Add-In Manager in SAS EG which is shown below in Image 8, is used for managing/adding new applications. The Temp Converter App and Titanic Explorer App, for example, can be added to the SAS EG using the Add-In Manager, for performing tasks such as temperature conversion and application of ML techniques on Titanic datasets respectively.

The 'SAI-Motor...' which is populated in Image 6 below, next to the 'Add-In Manager...', is the application/add-in which OCS Consulting built for one of its clients to analyze motor activity data in animals. The 'SAI-Motor…' stands for **S**AS Enterprise Guide **A**dd-**I**n for Motor Activity, in short SAI-Motor Activity. The add-in comprises a user interface built in Microsoft .NET Technology with a logical and statistical backend built in SAS. The add-in performs exploratory data analysis and repeated measures analysis using mixed modelling on motor activity data in animals.

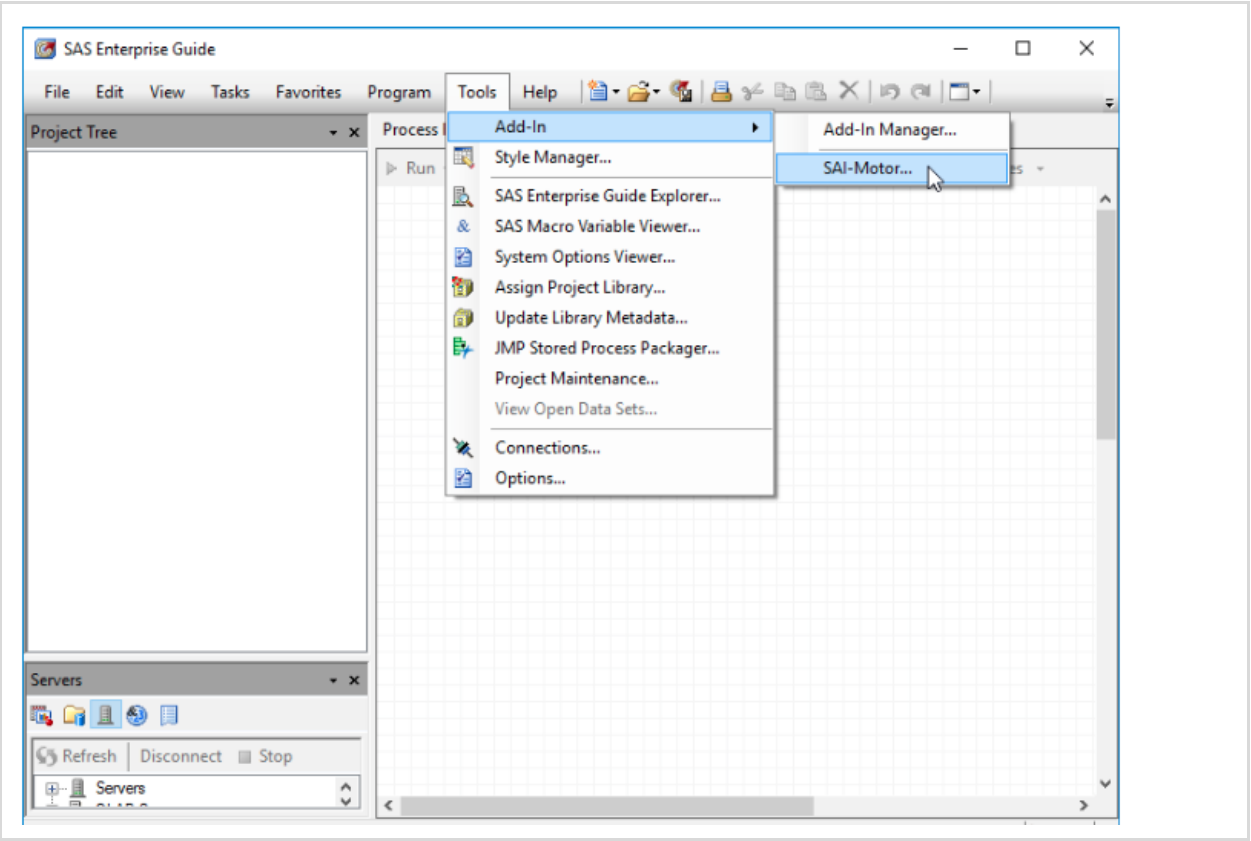



*Image 8*: The SAS Enterprise Guide Add-In Manager.

## CONCLUSION

By leveraging the power of the two-ends, i.e., the interface for user interaction (such as reading/viewing study data, preprocessing/cleaning and manipulating the processed data and generating reports for review or submission) and SAS for processing those requests at the backend, we can use SAS for our own needs. The purpose of building such an application would range from performing standard data management and analysis tasks to developing (training and testing of) sophisticated machine learning techniques for answering big questions on health, transportation and finance, etc.

The other advantage of such an approach is that the same SAS programs built for the backend of an application can easily be adapted and shipped to a completely different environment/platform without many changes (e.g., as a Web application). The application would then give the same service to users as before but now from a different platform.

## ACKNOWLEDGMENTS

I would like to thank my colleagues Jules van der Zalm and James Hunter for reviewing this paper and my colleagues Alan Purdom, James Hunter, Femke Sijtsma and Radoslaw Pszczolkowski for working with me on the development of the SAI-Motor Activity add-in application.

## CONTACT INFORMATION

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


| | |
|---|---|
| Author | Hailemichael M. Worku |
| Company | OCS Consulting |
| Address | Ruwekampweg 2G |
| City / Postcode | 's-Hertogenbosch / 5222 AT |
| Work Phone: | +31 (0)73 523 6000 |
| Email: | sasquestions@ocs-consulting.com |
| Web: | www.ocs-consulting.com/nl |