Paper CT02

Getting a Prompt Response Using ChatGPT and SAS

Stephen Mc Cawille, Daiichi-Sankyo, Munich, Germany

Abstract

This paper demonstrates the integration of ChatGPT, a state-of-the-art AI language model, with SAS® 9.4 using Application Programming Interface (API) keys and PROC HTTP to automate and simplify common programming tasks. By utilising SAS Macros, prompts related to code optimisation, debugging, and explanation are converted into API requests, enabling programmers to receive immediate feedback within SAS. These prompts are condensed into single-word macro calls (e.g., debug, optimise, explain), with flexible options such as *insert_error* and *insert_code* for further customisation. This integration could potentially enhance productivity and reduce the time spent on routine tasks, leading to improved outcomes in clinical programming.

Introduction

ChatGPT, developed by OpenAI, is an advanced AI language model capable of generating human-like responses to text-based prompts. This paper explores how ChatGPT can be integrated with SAS 9.4, a widely-used statistical software environment in the pharmaceutical industry, to assist programmers with tasks such as code debugging, optimisation, and explanation.

The goal is to demonstrate how SAS programmers can automate these tasks by sending API requests to ChatGPT using PROC HTTP, a SAS procedure that communicates with external web services. The API enables SAS to send prompts to ChatGPT and retrieve AI-generated responses, while PROC HTTP manages the transmission of these requests and responses over the web. Through this integration, programmers can receive real-time feedback from ChatGPT without leaving their SAS workflow, significantly enhancing efficiency.

By leveraging SAS Macros, this approach further simplifies interactions with ChatGPT, reducing the time spent on routine programming tasks. As automation continues to advance in the pharmaceutical industry—where precision and efficiency are critical—this integration offers a novel method for boosting productivity through the use of AI-driven tools in statistical programming.

Method

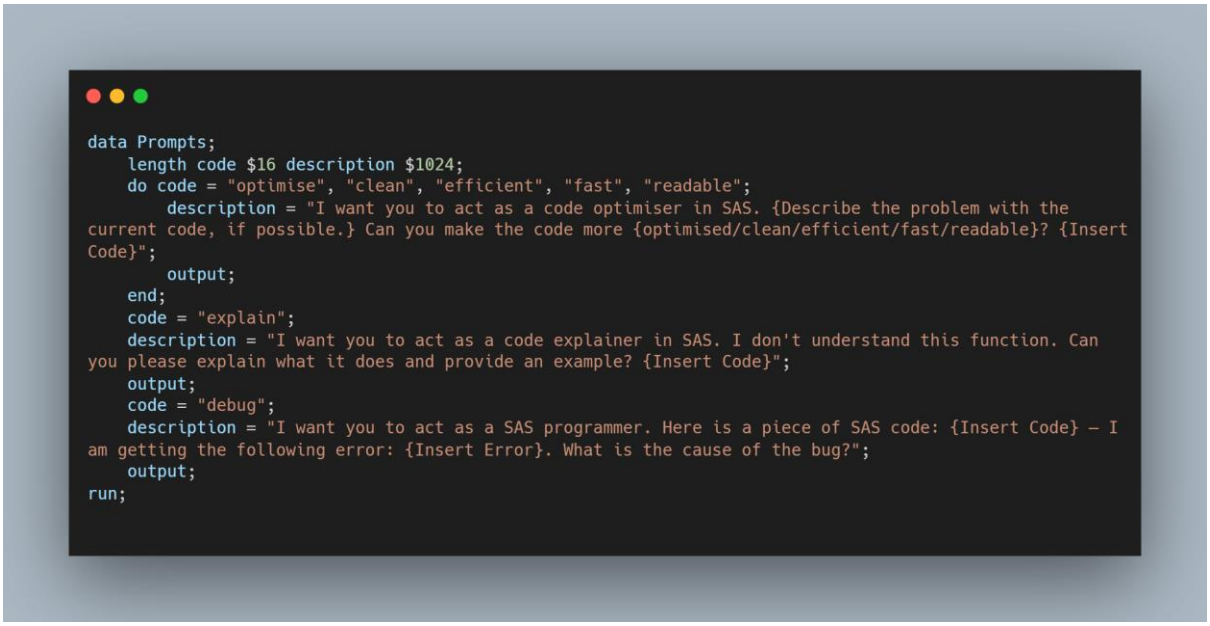
This section outlines the key steps involved in using the custom macro to integrate ChatGPT with SAS 9.4. The macro automates communication with the ChatGPT API for tasks such as code debugging, optimisation, and explanation. Each step includes SAS code and explanations of how it functions within the workflow.

Key Steps:

- *Prompts Dataset Setup*: Define a dataset that holds various prompts (e.g., optimise, debug, explain) for different actions that ChatGPT can handle.
- *API Key Reading*: Securely import the API key from a CSV file, which is used to authenticate requests to the ChatGPT API.
- *Core Processing: Error Checking, Prompt Setup and Validation*: Perform error checking and validate user inputs (e.g., the task type and code) to ensure that the correct action is selected from the Prompts dataset and properly formatted for ChatGPT.
- *Communication with OpenAI API*: Use PROC HTTP to send the formatted prompt to the ChatGPT API, then retrieve the AI-generated response for further use within SAS.
- *Displaying the Output*: Once the response is retrieved, format and display the output either as a report using PROC REPORT or as log output, depending on the user's preference. This allows the user to review the results directly in the SAS environment.

1. Prompts Dataset (Outside of the Macro)

Before using the macro, a Prompts dataset must be created. This dataset contains descriptions of various tasks (e.g., optimise, clean, debug) that the macro can perform based on user input. This dataset is stored outside the macro and accessed when needed.



```
data Prompts;
    length code $16 description $1024;
    do code = "optimise", "clean", "efficient", "fast", "readable";
        description = "I want you to act as a code optimiser in SAS. {Describe the problem with the
current code, if possible.} Can you make the code more {optimised/clean/efficient/fast/readable}? {Insert
Code}";
        output;
    end;
    code = "explain";
    description = "I want you to act as a code explainer in SAS. I don't understand this function. Can
you please explain what it does and provide an example? {Insert Code}";
    output;
    code = "debug";
    description = "I want you to act as a SAS programmer. Here is a piece of SAS code: {Insert Code} — I
am getting the following error: {Insert Error}. What is the cause of the bug?";
    output;
run;
```

The Prompts dataset defines different actions that the macro can trigger when communicating with the ChatGPT API, including tasks like optimise, explain, and debug. Each row in the dataset corresponds to a specific type of prompt (or task) that ChatGPT will respond to. The **code** variable holds the action's name, while the **description** variable provides the full text of the prompt.

- Placeholders: The prompt descriptions include placeholders such as {*Insert Code*} and {*Insert Error}*, which are dynamically replaced with user-provided inputs at runtime. These placeholders ensure the prompt is tailored to the specific request being made, allowing the macro to adapt to a wide range of programming tasks.
- Example Breakdown:
    - For the **optimise** action, the prompt is structured as:
      "I want you to act as a code optimiser in SAS. {Describe the problem with the current code, if possible.} Can you make the code more {optimised/clean/efficient/fast/readable}? {*Insert Code*}".
        - **{Describe the problem with the current code, if possible}**: This part allows the user to describe issues with the code, which gives ChatGPT additional context.
        - **{optimised/clean/efficient/fast/readable}**: This dynamic part lets the user specify what they want ChatGPT to focus on (e.g., optimisation or making the code more readable).
        - **{Insert Code}**: This placeholder is replaced by the user's actual code that needs optimisation.

2. API Key Reading (Outside of the Macro)

Next, the user's API key—used to authenticate and authorise access to the ChatGPT API—is read from a CSV file to ensure secure access for retrieving responses. The API key, typically starting with *sk-*, should be securely stored in a CSV file or through other methods like encrypted storage or environment variables. Keeping the API key separate from the main code helps minimise the risk of exposure, especially in shared or version-controlled environments. Although this process is demonstrated outside the macro, it can easily be implemented within the macro as well.

```
proc import datafile='API_Key.csv'
    out=api_key_dataset
    dbms=csv replace;
run;

data null;
    set api_key_dataset;
    call symputx('api_key', api_key);
run;
```

3. Core Processing: Error Checking, Prompt Setup and Validation

Now within the %chatgpt macro, the user's input is validated, and the prompt is prepared by dynamically replacing placeholders like {*Insert Code*} and {*Insert Error*} with user-provided code and error messages. This ensures the prompt is correctly formatted before being sent to the ChatGPT API.

3.1 Basic Error Checking

The macro first performs basic error checking to ensure that valid inputs are provided. This includes checking the presence of an API key, the validity of the print parameter, and confirming that the Prompts dataset exists.

```
%macro chatgpt(api_key=, print=true, out=, parameter=, insert_code=, describe_code=, insert_error=) / minoperator mindelimiter=' ';
    %let parameter = %superq(parameter);
    %let insert_code = %superq(insert_code);
    %let describe_code = %superq(describe_code);
    %let insert_error = %superq(insert_error);
    %let out = %upcase(&out.);
    %let print = %upcase(&print.);

    /* Error checking */
    %if(%bquote(&api_key.) =) %then %do;
        %put ERROR: No API key supplied.;
        %abort;
    %end;

    %if(NOT (&print. IN TRUE FALSE YES NO 1 0 LOG)) %then %do;
        %put ERROR: Invalid value for print. Expected TRUE, FALSE, YES, NO, 1, 0, or LOG.;
        %abort;
    %end;

    %if NOT %sysfunc(exist(work.Prompts)) %then %do;
        %put ERROR: Dataset work.Prompts does not exist!;
        %goto EndOfMacro;
    %end;

    %if %superq(PARAMETER) = %then %do;
        %put NOTE: The PARAMETER parameter allows only the following values:;
        options nosource;
        data _null_;
            set Prompts;
            by description notsorted;
            if first.description then put "NOTE- " @;
            put code @;
            if last.description then put " is for: " / @7 description/;
        run;
        options nosource;
        %goto EndOfMacro;
    %end;

    proc sql noprint;
        select code into :list_of_codes separated by " " from work.Prompts;
    quit;

    %if NOT (%superq(PARAMETER) in (&list_of_codes.)) %then %do;
        %put ERROR: The PARAMETER parameter has an invalid value!;
        %goto EndOfMacro;
    %end;
```

The first validation checks if an API key is provided. If missing, the macro displays an error and halts execution. This ensures secure access to the ChatGPT API.

- Note: If no API key is supplied, the macro outputs:
  "*ERROR: No API key supplied*.";

Next, the print parameter is validated to ensure it is set to one of the acceptable values (TRUE, FALSE, YES, NO, 1, 0, LOG). If the value is invalid, the macro raises an error and stops.

- Note: For invalid print values, the error message is:
  "*ERROR: Invalid value for print. Expected TRUE, FALSE, YES, NO, 1, 0, or LOG*.";

The macro checks that the Prompts dataset exists in the WORK library, which contains task templates for actions like optimise and debug. If the dataset is missing, the macro halts.

- Note: If the Prompts dataset is missing, the error is:
  "*ERROR: Dataset work.Prompts does not exist*!";

If the parameter is missing, the macro prints the valid task options from the Prompts dataset. If the user-provided parameter is invalid, PROC SQL retrieves the valid task codes, and the macro compares them against the input. If the parameter is invalid, the macro halts with an error.

- Note: If the parameter is invalid, the error is:
  "*ERROR: The PARAMETER parameter has an invalid value*!";

3.2 Prompt Setup

After passing basic error checking, the macro moves to building the prompt that will be sent to ChatGPT. The key objective of this section is to replace the placeholder text in the predefined task descriptions with user-specific inputs (such as code snippets and error messages).The code dynamically constructs the final prompt by replacing placeholders like {*Insert Code*}, {*Insert Error*}, and {*Describe problem with current code*} with the user's actual inputs.

```
data _null_;
    set Prompts;
    where code = symget('parameter');
    insert_code = symget('insert_code');
    insert_error = symget('insert_error');
    describe_code = symget('describe_code');
    length Prompt $ 32767;
    Prompt = prxchange('s/\{Describe problem with current code, if possible\.\}/'||strip(describe_code)||'/io', -1, description);
    Prompt = prxchange('s/\{Insert Code\}/'||strip(insert_code)||'/io', -1, Prompt);
    Prompt = prxchange('s/\{Insert Error\}/'||strip(insert_error)||'/io', -1, Prompt);
    Prompt = prxchange('s/\{optimize\/clean\/efficient\/fast\/readable\}/'||strip(code)||'/io', -1, Prompt);
    put "THE PROMPT:";
    put Prompt /;
run;
```

The data *_null_* step accesses the Prompts dataset, which contains these task templates (e.g., optimise, debug). The macro uses a where clause to match the user's selected task (specified in the parameter variable) to the relevant task code in the dataset, fetching the corresponding prompt description. The **symget()** function is used to retrieve the macro variable values for parameter, *insert_code*, *insert_error*, and *describe_code*. These values contain the user's inputs, such as the task type (e.g., optimise), the SAS code to be optimised or debugged, any relevant error messages, and additional context for ChatGPT. To ensure the prompt can handle long inputs, the length statement defines the maximum length of the final string (32767 characters).

The macro then uses the **prxchange()** function, which allows for regular expression-based replacements, to substitute the placeholders in the prompt with actual user inputs.

For instance, *{Describe problem with current code}* is replaced by **describe_code**, giving ChatGPT context for optimisation or debugging. Similarly, *{Insert Code}* is replaced by **insert_code**, which contains the user's SAS code, and *{Insert Error}* is replaced by **insert_error**, representing any errors the user wants ChatGPT to help debug. The placeholder *{optimised/clean/efficient/fast/readable}* is also dynamically replaced based on the task type selected by the user, indicating the specific aspect of the code (such as efficiency or readability) the user wants ChatGPT to focus on.Once the prompt has been fully constructed, it is displayed using the put statement, which outputs the formatted prompt to the log.

4. Communication with OpenAI API

After the prompt is constructed, the macro sends it to the ChatGPT API using PROC HTTP, facilitating communication between SAS and the OpenAI API. The macro builds a web request to send user inputs and receives the API's response for further processing within SAS.

```
filename in temp;
data _null_;
    file in;
    put "{";
    put '"model": "gpt-3.5-turbo", "messages": [{"role": "user", "content": "' "&parameter." '"},
        {"role": "user", "content": "' "&insert_code." '"}, {"role": "user", "content": "' "&insert_error." '"},
        {"role": "user", "content": "' "&describe_code." '"}]';
    put "}";
run;
```

The macro constructs a JSON-formatted body containing the model type (gpt-3.5-turbo), user task (parameter), code (*insert_code*), error (*insert_error*), and description (*describe_code*). These are all passed to the API in a format that ChatGPT can interpret.

- role is set to "user", which tells ChatGPT that this is input from the user.
- Each section of content is built from the user inputs and dynamically inserted into the JSON body.

Once the JSON body is created, the macro uses PROC HTTP to send this content to the ChatGPT API.

```
filename resp "%sysfunc(getoption(WORK))/response.json";

proc http
    method = "POST"
    url    = "https://api.openai.com/v1/chat/completions"
    ct     = "application/json"
    in     = in
    out    = resp;
    headers "Authorization" = "Bearer &api_key.";
run;
```

The macro defines:

- *in*: Temporary file containing the JSON request.
- *resp*: A file (*response.json*) that will store the API response.

The PROC HTTP statement executes an HTTP POST request to the OpenAI API endpoint. The JSON content is sent in the body of the request, and the response from the API is written to the response.json file. The API key is passed securely via the **Authorization** header using the **Bearer &api_key** format.

5. Displaying the Output

After receiving the response from the ChatGPT API, the macro formats and displays the output within the SAS environment, either as a report or log output, depending on the user's preference. This allows the user to easily review the AI-generated results directly in SAS.

```
libname response JSON fileref=resp;

data &outdata.;
    set response.choices_message;
    do row=1 to max(1,countw(content,'0A'x));
        outvar=scan(content,row,'0A'x);
        output;
    end;
    drop content;
run;
```

The response from the API, stored in *response.json*, is read as a JSON file using the *libname* statement with the JSON engine. This allows SAS to interpret the structure of the API response. In the following data step, the content field (which contains ChatGPT's response text) is processed and broken down into individual lines or rows using **scan()** and **countw()**, splitting the content at line breaks ('0A'x). The processed output is then stored in the specified dataset (*&outdata.*`), making it ready for further use.

Depending on the *print* parameter provided by the user, the macro either formats the output into a report or sends it directly to the SAS log.

- *Report Output (using PROC REPORT**):***

```
%if(&print. IN TRUE YES 1) %then %do;
    proc report data= &outdata.;
        column outvar;
        define outvar / display "" style(column)=[cellwidth=6in fontsize=10pt asis=ON];
    run;
%end;
```

If the user has requested a printed report (print=TRUE, YES, 1), the macro generates a detailed report using PROC REPORT. The output is formatted with custom styling (e.g., setting column width and font size) for better readability.

- *Log Output :*

```
%else %if(&print. = LOG) %then %do;
    data _null_;
        rc = jsonpp('resp','log');
    run;
%end;
```

If the user prefers to see the output in the SAS log (print=LOG), the macro uses **jsonpp()** (a built-in JSON pretty printer) to display the formatted response content directly in the log. This is useful for users who want quick access to the results without needing to generate a report.

6. Practical Examples

Below are examples of how the macro was applied, along with the corresponding outputs.

6.1 Code Explanation Example

In this example, the %chatgpt macro was used to explain the PROC FCMP procedure in SAS. The macro call, as shown in the screenshot below, resulted in a clear explanation of the procedure's purpose and usage. ChatGPT provided a detailed description of how PROC FCMP allows users to define and use custom functions within the SAS environment.

```
%chatgpt(api_key=&api_key.,print=true, out=, parameter= explain, insert_code={proc fcmp});
```

The PROC FCMP is a procedure in SAS (Statistical Analysis System) that is used to define and store user-defined functions in the SAS system. This procedure allows users to create custom functions and call them within SAS programs for data manipulation, analysis, and reporting. The FCMP procedure is commonly used in SAS to extend the capabilities of the built-in SAS functions and to perform complex calculations or transformations on data.
Users can define their own functions using PROC FCMP and specify input and output parameters, as well as the logic of the function. These custom functions can then be saved in SAS catalogs for reuse in different SAS programs or environments. PROC FCMP provides greater flexibility and control to SAS programmers by allowing them to create functions tailored to their specific needs and requirements.
Overall, PROC FCMP is a powerful tool in SAS programming for creating and managing custom functions, enhancing the functionality of SAS, and streamlining data analysis tasks.

Macro call for code explanation task and ChatGPT's detailed response explaining PROC FCMP.

6.2 Error Debugging Example

For the error debugging task, the macro was used to diagnose an issue with a missing dataset (WORK.RUN.DATA). As shown in the output screenshot, ChatGPT correctly identified the error and suggested multiple steps to resolve the issue, including verifying dataset existence, checking permissions, and correcting any typos in the dataset name.

```
%chatgpt(api_key=&api_key.,
         print=true,
         out=,
         parameter= debug,
         insert_code={data test123; set sashelp.class run;},
         insert_error=ERROR: File WORK.RUN.DATA does not exist.);
```

It seems that there is an error in your code. The error message "ERROR: File WORK.RUN.DATA does not exist." indicates that the dataset "RUN" does not exist in the WORK library.
To debug this issue, you can check the following:
- Ensure that the dataset name is correct. In your code, you are trying to reference a dataset named "RUN". Make sure that this dataset exists in the WORK library or the library you are referencing.
- Check for any typos or inconsistencies in the dataset name.
- Verify that you have the necessary permissions to access the dataset.
You may want to correct the dataset name or access the correct dataset in order to resolve this error.

Macro call for error debugging task and ChatGPT's response diagnosing the missing dataset issue

6.3 Code Efficiency Example

In this task, the %chatgpt macro was used with the **"efficient"** parameter to improve the performance of a slow-running data step. ChatGPT provided several recommendations aimed at increasing efficiency, such as using smaller datasets, applying WHERE statements to filter only necessary rows, and avoiding unnecessary calculations. These suggestions, as shown in the screenshot, are specifically focused on improving the efficiency of the data step in SAS programming.

```
%chatgpt(api_key=&api_key.,
         print=true,
         out=,
         parameter= efficient,
         insert_code={data test123; set sashelp.class; run;},
         describe_code= That data step works slow.);
```

To improve the efficiency of your data step, you can consider the following tips:
1. Use a smaller dataset: If you do not need to process all the observations in the dataset, consider subsetting the data before performing any operations.
2. Use WHERE statement: Instead of processing all observations in the dataset, use a WHERE statement to filter only the necessary rows.
3. Use INDEX or BY statement: When sorting or merging datasets, using an INDEX or BY statement can improve processing time.
4. Avoid unnecessary calculations: Only include variables and calculations that are necessary for your analysis to reduce processing time.
5. Use appropriate functions: Choose appropriate functions and data step techniques that are optimized for the task at hand.
6. Check for syntax errors: Make sure the code is error-free to prevent unnecessary processing delays.
By implementing these tips, you can make your data step more efficient and reduce processing time.

Macro call for the efficient parameter and ChatGPT's suggestions for improving data step efficiency.

7. Limitations and Future Work

Although the integration of ChatGPT with SAS has proven effective for automating routine programming tasks, certain limitations affect its suitability for continuous workflows. One key limitation is the lack of **continuous code monitoring**. Unlike Aider, an AI-powered coding assistant that provides real-time feedback during code execution and monitors logs, %chatgpt relies on isolated API calls. This makes %chatgpt less effective for dynamic workflows where continuous feedback is essential for detecting and fixing errors in real-time. Aider, for instance, allows ongoing interaction, suggesting fixes as the code runs, whereas %chatgpt processes only one request at a time. Additionally, %chatgpt handles each request independently, without any awareness of the broader project context, limiting its ability to manage complex, multi-file changes. In contrast, Aider can handle multiple files simultaneously by providing the AI with a map of the entire Git repository, making it more suitable for large-scale development projects.

Another limitation is that ChatGPT's feedback may sometimes lack precision or context-specific relevance, especially in specialised programming domains. While ChatGPT generally provides helpful and accurate suggestions, certain niche or complex issues may require manual validation to ensure the recommendations are suitable for the task at hand.

To address these limitations, several future enhancements could be explored. First, implementing **real-time log monitoring** would enable %chatgpt to dynamically suggest fixes during code execution, similar to Aider's continuous feedback capabilities, making it more effective for real-time workflows. Enhancing %chatgpt's ability to handle multi-file awareness would also improve its capacity to manage complex workflows. By enabling the macro to handle multiple files simultaneously, similar to Aider's integration with Git, %chatgpt could handle multi-file edits more smoothly and efficiently. Another valuable improvement would be to allow for **iterative dialogue** with ChatGPT, enabling users to ask follow-up questions during debugging or optimisation processes. This would create a more dynamic interaction and help users gain deeper insights into problem-solving.

Lastly, exploring the potential for offline or locally deployed models, such as fine-tuned large language models (LLMs) within the SAS environment, could reduce reliance on API calls, lower costs, and enhance security for users handling sensitive or confidential data.

8. Conclusion

Integrating ChatGPT with SAS has shown great potential for automating routine tasks such as code optimisation, debugging, and explanation. This collaboration allows programmers to streamline workflows and focus on more complex analytical work, which is especially valuable in industries like pharmaceuticals where precision and efficiency are critical.Despite its benefits, limitations such as the lack of continuous monitoring, built-in version control, and the occasional need for manual validation of ChatGPT's suggestions highlight areas for improvement. However, the productivity gains from automating common tasks demonstrate the utility of this integration.

Future enhancements, including real-time log monitoring, multi-file awareness, iterative dialogue, and offline models, could further improve the tool's capabilities. These developments would enable %chatgpt to better handle more complex projects and provide continuous, context-aware feedback.In summary, while some challenges remain, the integration of ChatGPT with SAS is a promising tool for improving efficiency in programming workflows, with strong potential for future growth and refinement.

REFERENCES

[1] ListenData. Run ChatGPT inside SAS. ListenData. 2023. Available from: https://www.listendata.com/2023/03/run-chatgpt-inside-sas.html

[2] Gauthier P. Aider: GPT powered coding in your terminal. GitHub. 2023. Available from: https://github.com/paul-gauthier/aider

[3] Analytium. GenAI SAS programming demo. Analytium. 2023. Available from: https://analytium.com/resources/genai-sas-programming-demo

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Daiichi-Sankyo for providing me with the opportunity to attend this conference and present my work. I also extend my thanks to Antonio Lovatin and Ruth Belén Alonso Meseguer for their guidance throughout the development of this paper. Special thanks to Stu Sztukowski and, in particular, Bartosz Jabłoński for their invaluable help with the initial code structure.

RECOMMENDED READING

For further details on the macro implementation and to access the full code, visit: https://github.com/smccawille88/TEST/blob/main/CHATGPT.sas.

CONTACT INFORMATION

Stephen Mc Cawille
Daiichi-Sankyo Europe
Registered Office: Daiichi Sankyo Europe. GmbH Zielstattstr. 48 81379 Munich Germany
Email: stephen.mccawille@daiichi-sankyo.eu
Web: www.daiichi-sankyo.eu