

PharmaSUG 2026 - Paper AI-432

Evaluation of Azure OpenAI ChatGPT API as Code Assistance Tools for Statistical Programming in SAS, R and Python.

Ajay Gupta, Daiichi Sankyo
Misikir Tesfaye, Daiichi Sankyo

Abstract

Statistical programming in pharmaceutical research requires high levels of accuracy, responsibility and traceability. Recent advances in Generative Artificial Intelligence(GenAI) have introduced large language models capable of assisting with code generation through application programming interfaces(APIs).This paper presents a structured evaluation of the ChatGPT model accessed via the Azure OpenAI as a code assistance tool for statistical programming in SAS,R and Python.

The evaluation was conducted across three execution platforms, using six predefined prompts grouped by task complexity, ranging from dummy data creation to reuse of macro and function generations. The same prompts group were applied consistently across platforms. A total of 18 language-platform execution scenarios were evaluated. All code generated by ChatGPT was executed without manual modification. Successful execution was achieved in 100% test cases, with correct output generation and creation of corresponding log files in user-specified directories.

These findings are consistent with previously published peer-reviewed studies that report high rates of synthetically correct code generation by large language models under controlled prompt conditions (Chen, M. (2021)). This result suggests that API-based GenAI can serve as a reliable code assistance tool for statistical programming workflows when used with appropriate human oversight and governance control.

Introduction

Statistical programming is a critical component of pharmaceutical research ,supporting clinical trials analysis, regulatory submissions, and explanatory data analyses and python are widely used due to their complementary strengths. However, development of statistical code often involves repetitive patterns, boilerplate logics and manual translation across languages.

Generative Artificial Intelligence(GenAI),particularly large language models accessed via APIs, has emerged as a potential productivity – enhancing tools (Achiam et al., 2023; Nijkamp, 2022). Unlike interactive user interfaces, API-based usage enables reproducibility, traceability and controlled integration, which are essential in regulated environments.

This paper evaluates the use of the Azure OpenAI ChatGPT API as a code assistant tool within enterprise-grade execution environment.

Objective

The objective of the evaluation was to:

- Assess the ability of ChatGPT(via Azure OpenAI API) to generate executables SAS, R and Python code.
- Evaluate code correctness across six structured prompts.
- Ensure reproducibility through consistent prompt reuse across platforms

- Validate execution and log generation in SAS studio and Databricks for R and Python.
- Examine the role of prompt engineering in code quality.

Methods

1. System Architecture

The evaluation of architecture consists of the following:

- Azure OpenAI Service – ChatGPT(GPT-4o deployment)accessed via REST API.
- SAS Studio Execution environment for SAS programs.
- Databricks – Execution environment for R and Python code.

All GenAI interaction performed programmatically, and no proprietary or sensitive data were used.

2. Prompt Grouping and Test Design

Six prompts were defined and organized into three functional groups. Each group was executed independently on each platform, using the same logical task definition.

| Group | Prompt Ids | Description |
|---------|------------|--|
| Group 1 | P1 – P2 | Dummy data creation and transformation |
| Group 2 | P3 – P4 | Statistical programming using R function and Macro |
| Group 3 | P5 – p6 | Statistical analysis and reporting and reusable code generation. |

3. Prompt definitions

- P1 : Write SAS code to programmatically generate a dummy clinical dataset.
- P2 : with variables: Age Gender Treatment Time Event Do not use DATALINES or CARDS. Use loops and random number function.
- P3: Write SAS code to summarize Age by Treatment.
- P4: Create SAS macro that produces summary statistics for any numeric variables by Treatment.
- P5: Provide SAS code for survival analysis(time-to-event) with strata by Treatment.
- P6: Generate a Kaplan-Meier survival plot in SAS.

Each prompt explicitly specified in programming languages, execution environment and output and log location.


```

%chatgpt(prompt=&full_prompt);

libname respjson json fileref=resp;
title "ChatGPT Response to Create Dummy Dataset";
proc print data =respjson.choices_message;run;

data _null_;
set respjson.choices_message;
file "&outpath/generated_SAS_Code1.sas";
put content;
run;

proc printto log="&outpath/generated_code1.log" new;
run;

/** ChatGPT response Code Generated for prompt P1 & P2***/
Here's how you can generate a dummy clinical dataset programmatically using SAS. It
incorporates Step 1 and Step 2 by using loops and random number functions to create variables
`ID`, `Age`, `Gender`, `Treatment`, `Time`, and `Event`.

### SAS Code to Generate Dummy Clinical Dataset:

```sas
* Step 1: Generate a dummy clinical dataset programmatically;
data clinical_data;
 call streaminit(123); /* Initialize the random number generator for reproducibility */

 * Generate data for 100 observations;
 do ID = 1 to 100;

 * Generate random Age between 30 and 80;
 Age = rand("Integer", 30, 80);

 * Generate random Gender (1 = Male, 2 = Female);
 Gender = rand("Integer", 1, 2);

 * Assign random treatment group (1 = Control, 2 = Treatment A, 3 = Treatment B);
 Treatment = rand("Integer", 1, 3);

```

```

 * Generate random Time (event time) between 0 and 10 years;
 Time = round(rand("Uniform") * 10, 0.01); /* Uniform distribution between 0 and 10
years */

 * Simulate Event (0 = no event, 1 = event occurs);
 Event = rand("Bernoulli", 0.5); /* Assume a 50% probability of the event occurring */

 * Output the observation;
 output;
 end;
run;

* Step 2: Print and check the generated dataset (optional);
proc print data=clinical_data(obs=20); /* Display first 20 observations */
run;

proc freq data=clinical_data;
 tables Gender Treatment Event; /* Frequency tables for categorical variables */
run;

proc means data=clinical_data;
 var Age Time; /* Summary statistics for continuous variables */
run;
...

Explanation:
1. Data Step: The dataset is created using a `do` loop to simulate 100 patients (`ID` ranging from 1 to 100).
2. Random Number Functions:
 - Age: Generated using the `rand("Integer", min, max)` function for integer values between 30 and 80.
 - Gender: Binary random value (1 for Male, 2 for Female).
 - Treatment: A categorical variable with three levels (1 = Control, 2 = Treatment A, 3 = Treatment B).
 - Time: Continuous values generated using the `rand("Uniform")` function scaled between 0 and 10 years.
 - Event: A binary variable simulated using the Bernoulli distribution (`rand("Bernoulli", probability)`), where the probability of an event occurring is set to 50%.
3. Reproducibility: The `call streaminit(123)` function is used to initialize the random number generator, ensuring that the results are reproducible.

```

4. **Output and Review**: The dataset can be visualized using `PROC PRINT`, summarized using `PROC FREQ` for categorical variables, and `PROC MEANS` for continuous variables.

You can modify the `do` loop limit to adjust the number of observations or tweak the random number parameters to create variations in the dataset's structure!

```
/** SAS Code Prompt P3 and P4**/
%let api_key=xxxxxxxxxxxxxxxxxxxxxxxxx;
%let url = yyyyyyyyyyyyyyyyyy;
%let outpath = zzzzzzzzzzzzzzzzzzzzzzzzz;

%macro chatgpt(prompt=);
 Filename in temp;
 /*Create JSON payload for ChatGPT API */
 Data _null_;
 file in;
 put '{';
 put ' "messages": [';
 put ' {"role": "system", "content": "You are a helpful assistant."},';
 put ' {"role": "user", "content": "' &prompt" '}'';
 put ']';
 put '}';
 run;
 /*Send POST request to ChatGPT*/
 Filename resp "response.json";
 Proc http
 url="&url."
 method="POST"
 in=in
 out=resp;
 headers
 "Content-Type"="application/json"
 "Authorization" = "&api_key.";
 run;
%mend chatgpt;

%let p1 = Step1: Write SAS code to programatically generate a dummy clinical dataset;
%let p2 = Step2: with variables:ID Age Gender Treatment Time Event and Donot use DATALINES or
CARDS.Use loops and random number function;
%let p3 = Step3: Write SAS code to summarize Age by Treatment;
```

```

%let p4 = Step4: Create SAS macro that produces summary statistics for any numeric variables
by Treatment;
%let full_prompt= &p1 &p3 &p4;
%chatgpt(prompt=&full_prompt);
libname respjson json fileref=resp;
title "ChatGPT Response to Use the created Dataset to make summary tables and macros";
proc print data =respjson.choices_message;run;
data _null_;
set respjson.choices_message;
file "&outpath/generated_SAS_Code2.sas";
put content;
run;
proc printto log="&outpath/generated_code2.log" new;
run;

```

**/\*\* ChatGPT response Code Generated for P3 & P4\*\*/**

Below is how you can approach Step 3 (writing SAS code to summarize Age by Treatment) and Step 4 (creating a SAS macro for summary statistics of any numeric variables by Treatment). For Step 1, you can assume a dummy `ClinicalData` dataset, or you can adapt it to existing clinical datasets you may have.

---

**### Step 1: Sample Dummy Clinical Data**

Here's a simple dummy dataset with `Age`, `Gender`, and `Treatment` variables:

```

````SAS
data ClinicalData;
  input ID Age Gender $ Treatment $;
  datalines;
  1 34 M A
  2 29 F B
  3 45 M A
  4 50 F B
  5 38 M A
  6 47 F B
;
run;
````

```

---

### ### Step 3: Summarize Age by Treatment

To summarize `Age` by `Treatment`, you can use the `PROC MEANS` procedure. Here's the SAS code:

```
```SAS
proc means data=ClinicalData mean std min max;
  class Treatment;
  var Age;
run;
```
```

This code:

- Uses `PROC MEANS` to calculate summary statistics (mean, standard deviation, minimum, and maximum).
- Groups the results by `Treatment`.
- Displays statistics for the `Age` variable.

---

### ### Step 4: Create SAS Macro for Summary Statistics

To make the code more dynamic and usable for any numeric variable by `Treatment`, you can create a SAS macro:

```
```SAS
%macro SummarizeByTreatment(dataset, var);
  proc means data=&dataset mean std min max;
    class Treatment;
    var &var;
  run;
%mend SummarizeByTreatment;
```
```

#### #### Example Macro Usage:

If you wanted to summarize the `Age` variable:

```
```SAS
%SummarizeByTreatment(ClinicalData, Age);
```
```

If your dataset had additional numeric variables (e.g., `Height`, `Weight`, etc.), you could call the macro for those variables as well:

```
``SAS
%SummarizeByTreatment(ClinicalData, Height);
%SummarizeByTreatment(ClinicalData, Weight);

```

This macro:

- Makes the code reusable for any numeric variable.
- Accepts a dataset name (`dataset`) and variable name (`var`) as parameters.

---

### Summary

- Step 3 provides code to summarize `Age` by `Treatment` using `PROC MEANS`.
- Step 4 introduces a flexible SAS macro (`SummarizeByTreatment`) for summarizing any numeric variables by `Treatment`.

You can test and expand this code for your specific data analysis needs. Let me know if you need further clarification!

```
/** SAS Code Prompt P5 and P6**/
```

```
%let api_key=xxxxxxxxxxxxxxxxxxxx;
%let url = yyyyyyyyyyyyyyyyyyyyyyy;
%let outpath = zzzzzzzzzzzzzzzzzzz;
```

```
%macro chatgpt(prompt=);
```

```
Filename in temp;
```

```
/*Create JSON payload for ChatGPT API */
```

```
Data _null_;
```

```
file in;
```

```
put '{';
```

```
put ' "messages": [';
```

```
put ' {"role": "system", "content": "You are a helpful assistant."},';
```

```
put ' {"role": "user", "content": "' &prompt" '}'';
```

```
put ']';
```

```
put '}';
```

```
run;
```

```
/*Send POST request to ChatGPT*/
```

```
Filename resp "response.json";
```

```

Proc http
url="&url."
method="POST"
in=in
out=resp;
headers
"Content-Type"="application/json"
"Authorization" = "&api_key.";
run;
%mend chatgpt;

%let p1 = Step1: Write SAS code to programatically generate a dummy clinical dataset;
%let p2 = Step2: with variables:ID Age Gender Treatment Time Event and donot use DATALINES or CARDS.Use loops and random number function;
%let p3 = Step3: Provide SAS code for survival analysis(time-to-event) with strata by Treatment;
%let p4 = Step4: Generate a Kaplan-Meier survival plot in SAS;

%let full_prompt= &p1 &p2 &p3 &p4 ;

%chatgpt(prompt=&full_prompt);

libname respjson json fileref=resp;
title "ChatGPT Response to use the created dataset and make survival plots and tables";
proc print data =respjson.choices_message;run;

data _null_;
set respjson.choices_message;
file "&outpath/generated_SAS_Code3.sas";
put content;
run;
proc printto log="&outpath/generated_code3.log" new;
run;
/** ChatGPT response Code Generated for P5 & P6***/
Below is how you can approach Step 3 (writing SAS code to summarize Age by Treatment) and Step 4 (creating a SAS macro for summary statistics of any numeric variables by Treatment). For Step 1, you can assume a dummy `ClinicalData` dataset, or you can adapt it to existing clinical datasets you may have.

Step 1: Sample Dummy Clinical Data
Here's a simple dummy dataset with `Age`, `Gender`, and `Treatment` variables:

```

```

```SAS
data ClinicalData;
    input ID Age Gender $ Treatment $;
    datalines;
    1 34 M A
    2 29 F B
    3 45 M A
    4 50 F B
    5 38 M A
    6 47 F B
;
run;
```

```

### ### Step 3: Summarize Age by Treatment

To summarize `Age` by `Treatment`, you can use the `PROC MEANS` procedure. Here's the SAS code:

```

```SAS
proc means data=ClinicalData mean std min max;
    class Treatment;
    var Age;
run;
```

```

This code:

- Uses `PROC MEANS` to calculate summary statistics (mean, standard deviation, minimum, and maximum).
- Groups the results by `Treatment`.
- Displays statistics for the `Age` variable.

---

### ### Step 4: Create SAS Macro for Summary Statistics

To make the code more dynamic and usable for any numeric variable by `Treatment`, you can create a SAS macro:

```

```SAS
%macro SummarizeByTreatment(dataset, var);
    proc means data=&dataset mean std min max;
        class Treatment;
        var &var;
    run;
%mend SummarizeByTreatment;
```

```

```

...
Example Macro Usage:
If you wanted to summarize the `Age` variable:
```SAS
%SummarizeByTreatment(ClinicalData, Age);
...

If your dataset had additional numeric variables (e.g., `Height`, `Weight`, etc.), you could
call the macro for those variables as well:
```SAS
%SummarizeByTreatment(ClinicalData, Height);
%SummarizeByTreatment(ClinicalData, Weight);
...

This macro:
- Makes the code reusable for any numeric variable.
- Accepts a dataset name (`dataset`) and variable name (`var`) as parameters.

Summary
- Step 3 provides code to summarize `Age` by `Treatment` using `PROC MEANS`.
- Step 4 introduces a flexible SAS macro (`SummarizeByTreatment`) for summarizing any numeric
variables by `Treatment`.

You can test and expand this code for your specific data analysis needs. Let me know if you
need further clarification!

```

```

/** R Code Prompt P1 and P2***/
%r
Step 1: Install the necessary Libraries
install.packages("httr")
install.packages("jsonlite")
library(httr)
library(jsonlite)

Step 2: Set API Key and URL
api_key<-"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
url<-"yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy"
outputpath<-"zzzzzzzzzzzzzzzzzzzzzzzzzzzz "

Step 3: Create output directory if it doesnot exist
dir.create(outputpath,recursive=TRUE ,showWarnings=FALSE)

```

```

Step 4: Define the Prompts
p1<-"Step1: Write R code to programatically generate a dummy clinical dataset."
p2<-"Step2: with variables:ID Age Gender Treatment Time Event.Use loops and random number
function."

full_prompt<-paste(p1,p2)

Step 5: Construct JSON payload
payload<-list(
 model="gpt-4o",
 messages=list(
 list(role="system",content="You are a helpful assistant."),
 list(role="user",content=full_prompt)
)
)

Step 6: Send post request to OpenAI
response<-POST(
 url=url,
 add_headers(
 Authorization=paste("Bearer",api_key),
 `Content-Type`="application/json"
),
 body=toJSON(payload,auto_unbox=TRUE),
 encode="json"
)

raw_json<-content(response,as="text",encoding="UTF-8")
parsed<-fromJSON(raw_json)

if(!is.null(parsed$choices) &&
length(parsed$choices) >= 1 &&
!is.null(parsed$choices$message$content)) {
 generated_code01<-as.character(parsed$choices$message$content)
 cat("\n Successfully extracted generated R code.\n")

 # Step 9: Save generated code to .R and .log file
r_file_path<-file.path(outpath,"generated_code01.R")
log_file_path<-file.path(outpath,"chatgpt_code_generation01.log")

```

```

writeLines(generated_code01,con = r_file_path)
cat("\n R code saved to:",r_file_path)

Step 10: Write log file

writeLines(
 c(
 paste("Timestamp:", Sys.time()),
 "prompt sent to ChatGPT:",
 full_prompt,
 "-----",
 "Generated R Code:",
 generated_code01
),
 con = log_file_path
)
cat("\n Log file saved to :", log_file_path)
} else {
 cat("\n No content returned from ChatGPT.\n")
 cat(raw_json)
}

/** Python Code for prompt P1 & P2**/
%python
Step 1: Load the necessary libraries
import os
import requests
import json
from datetime import datetime

Step 2: Set up variables
api_key = "xxxxxxxxxxxxxxxx"
url = "yyyyy"
outpath = "zzzzzzzzzzzzzzzzzzzz"

os.makedirs(outpath, exist_ok=True)

Step 3: Prompts set up

```

```

p1 = "Step1: Write Python code to programatically generate a dummy clinical dataset."
p2 = "Step2: with variables:ID Age Gender Treatment Time Event.Use loops and random number
function."

full_prompt = "\n".join([p1,p2])

Step 4: Prepare JSON payload for ChatGPT
payload = {
 "model": "gpt-4o",
 "messages":[
 {"role": "system", "content": "You are a helpful assistant."},
 {"role": "user", "content": full_prompt}
]
}

headers = {
 "Authorization": f"Bearer {api_key}",
 "Content-Type": "application/json"
}

Step 5: Send a POST request
response = requests.post(url, headers=headers, json=payload)

Step 7: Parse and extract the response
if response.status_code != 200:
 raise Exception(f"ChatGPT API request failed with status code: {response.status_code}")

data = response.json()
try:
 generated_code01 = data["choices"][0]["message"]["content"]
 print("Successfully extracted python code from ChatGPT response")
except (KeyError, IndexError):
 print("Failed to extract generated code. Raw response.")
 print(json.dumps(data, indent=2))
 raise

Step 8: Save the generated code and log to a file
py_file_path = os.path.join(outpath, "generated_code01.py")
log_file_path = os.path.join(outpath, "chatGPT_code_generation01.txt")

```

```

with open(py_file_path, "w") as py_file:
 py_file.write(generated_code01)
print("Python code saved to:", py_file_path)

with open(log_file_path, "w") as log_file:
 log_file.write(f"Timestamp: {datetime.now()}\n")
 log_file.write(f"Prompt sent to chatGPT:\n")
 log_file.write(full_prompt + "\n")
 log_file.write("-----\n")
 log_file.write(f"Generated Python Code:\n")
 log_file.write(generated_code01)
print(f"Log saved to:{log_file_path}")

/**** R Code Prompt P3 and P4****/
%r
Step 1: Install the necessary Libraries
install.packages("httr")
install.packages("jsonlite")
library(httr)
library(jsonlite)

Step 2: Set API Key and URL
api_key<-"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
url<-"yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy"
outpath<-"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ "

Step 3: Create output directory if it doesnot exist
dir.create(outpath,recursive=TRUE ,showWarnings=FALSE)

Step 4: Define the Prompts
p1 <- "Step1: Write R code to programatically generate a dummy clinical dataset."
p2 <- "Step2: with variables:ID Age Gender Treatment Time Event.Use loops and random number function."
p3 <- "Step3: Write R code to summarize Age by Treatment."
p4 <- "Step4: Create R function that produces summary statistics for any numeric variables by Treatment."

prompts<-c(p1,p2,p3,p4)
full_prompt<-paste(prompts, collapse = "\n\n")

```

```

Step 5: Construct JSON payload
payload<-list(
 model="gpt-4o",
 messages=list(
 list(role="system",content="You are a helpful assistant."),
 list(role="user",content=full_prompt)
)
)

Step 6: Send post request to OpenAI
response<-POST(
 url=url,
 add_headers(
 Authorization=paste("Bearer",api_key),
 `Content-Type`="application/json"
),
 body=toJSON(payload,auto_unbox=TRUE),
 encode="json"
)

raw_json<-content(response,as="text",encoding="UTF-8")
parsed<-fromJSON(raw_json)

if(!is.null(parsed$choices) &&
length(parsed$choices) >= 1 &&
!is.null(parsed$choices$message$content)) {
 generated_code02<-as.character(parsed$choices$message$content)
 cat("\n Successfully extracted generated R code.\n")

 # Step 9: Save generated code to .R and .log file
 r_file_path<-file.path(outpath,"generated_code02.R")
 log_file_path<-file.path(outpath,"chatgpt_code_generation02.log")

 writeLines(generated_code02,con = r_file_path)
 cat("\n R code saved to:",r_file_path)

Step 10: Write log file

writeLines(

```



```

full_prompt = "\n".join([p1,p2,p3,p4])

Step 4: Prepare JSON payload for ChatGPT
payload = {
 "model": "gpt-4o",
 "messages":[
 {"role": "system", "content": "You are a helpful assistant."},
 {"role": "user", "content": full_prompt}
]
}

headers = {
 "Authorization": f"Bearer {api_key}",
 "Content-Type": "application/json"
}

Step 5: Send a POST request
response = requests.post(url, headers=headers, json=payload)

Step 7: Parse and extract the response
if response.status_code != 200:
 raise Exception(f"ChatGPT API request failed with status code: {response.status_code}")

data = response.json()
try:
 generated_code02 = data["choices"][0]["message"]["content"]
 print("Successfully extracted python code from ChatGPT response")
except (KeyError, IndexError):
 print("Failed to extract generated code. Raw response.")
 print(json.dumps(data, indent=2))
 raise

Step 8: Save the generated code and log to a file
py_file_path = os.path.join(outpath, "generated_code02.py")
log_file_path = os.path.join(outpath, "chatGPT_code_generation02.txt")

with open(py_file_path, "w") as py_file:
 py_file.write(generated_code02)
print("Python code saved to:", py_file_path)

```

```

with open(log_file_path, "w") as log_file:
 log_file.write(f"Timestamp: {datetime.now()}\n")
 log_file.write(f"Prompt sent to chatGPT:\n")
 log_file.write(full_prompt + "\n")
 log_file.write("-----\n")
 log_file.write(f"Generated Python Code:\n")
 log_file.write(generated_code02)
print(f"Log saved to:{log_file_path}")

/** R Code Prompt P5 and P6**/
%r
Step 1: Install the necessary Libraries
install.packages("httr")
install.packages("jsonlite")
library(httr)
library(jsonlite)

Step 2: Set API Key and URL
api_key<-"xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
url<-"yyyyyyyyyyyyyyyyyyyyyyyyyyyy"
outpath<-"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"

Step 3: Create output directory if it doesnot exist
dir.create(outpath,recursive=TRUE ,showWarnings=FALSE)

Step 4: Define the Prompts
p1 <- "Step1: Write R code to programatically generate a dummy clinical dataset"
p2 <- "Step2: with variables:ID Age Gender Treatment Time Event.Use loops and random number function"
p3 <- "Step3: Provide R code for survival analysis(time-to-event) with strata by Treatment"
p4 <- "Step4: Generate a Kaplan-Meier survival plot in R"

prompts<-c(p1,p2,p3,p4)
full_prompt<-paste(prompts, collapse = "\n\n")

Step 5: Construct JSON payload
payload<-list(
 model="gpt-4o",


```

```

messages=list(
 list(role="system",content="You are a helpful assistant."),
 list(role="user",content=full_prompt)
)
)

Step 6: Send post request to OpenAI
response<-POST(
 url=url,
 add_headers(
 Authorization=paste("Bearer",api_key),
 `Content-Type`="application/json"
),
 body=toJSON(payload,auto_unbox=TRUE),
 encode="json"
)

raw_json<-content(response,as="text",encoding="UTF-8")
parsed<-fromJSON(raw_json)

if(!is.null(parsed$choices) &&
length(parsed$choices) >= 1 &&
!is.null(parsed$choices$message$content)) {
 generated_code03<-as.character(parsed$choices$message$content)
 cat("\n Successfully extracted generated R code.\n")

 # Step 9: Save generated code to .R and .log file
 r_file_path<-file.path(outpath,"generated_code03.R")
 log_file_path<-file.path(outpath,"chatgpt_code_generation03.log")

 writeLines(generated_code03,con = r_file_path)
 cat("\n R code saved to:",r_file_path)

Step 10: Write log file

writeLines(
 c(
 paste("Timestamp:", Sys.time()),
 "prompt sent to ChatGPT:",

```



```

 "model": "gpt-4o",
 "messages":[
 {"role": "system", "content": "You are a helpful assistant."},
 {"role": "user", "content": full_prompt}
]
}
headers = {
 "Authorization": f"Bearer {api_key}",
 "Content-Type": "application/json"
}

Step 5: Send a POST request
response = requests.post(url, headers=headers, json=payload)

Step 7: Parse and extract the response
if response.status_code != 200:
 raise Exception(f"ChatGPT API request failed with status code: {response.status_code}")

data = response.json()
try:
 generated_code03 = data["choices"][0]["message"]["content"]
 print("Successfully extracted python code from ChatGPT response")
except (KeyError, IndexError):
 print("Failed to extract generated code. Raw response.")
 print(json.dumps(data, indent=2))
 raise

Step 8: Save the generated code and log to a file
py_file_path = os.path.join(outpath, "generated_code03.py")
log_file_path = os.path.join(outpath, "chatGPT_code_generation03.txt")

with open(py_file_path, "w") as py_file:
 py_file.write(generated_code03)
print("Python code saved to:", py_file_path)

with open(log_file_path, "w") as log_file:
 log_file.write(f"Timestamp: {datetime.now()}\n")
 log_file.write(f"Prompt sent to chatGPT:\n")
 log_file.write(full_prompt + "\n")

```

```
log_file.write("-----\n")
log_file.write(f"Generated Python Code:\n")
log_file.write(generated_code03)
print(f"Log saved to:{log_file_path}")
```

## Result

All the 18-one-language-platform execution scenarios were completed successfully, resulting in 100% execution success rate. No syntax related errors were observed across any of the SAS,R and Python code generated. In all cases, execution logs were successfully created and written to the user-specified directories within SAS studio and Databricks environments. The output generated consistently matched the expected data structures and analytical results defined in the evaluation criteria. Additionally, reuse of the same prompts groups across multiple platforms produced consistent and reproducible outcomes, demonstrating the effectiveness of the standardized prompt design in minimizing variability.

## Limitation

Performance optimization was not evaluated, as execution efficiency depends on configuration, data volume and validation requirements, which were outside the scope of this controlled environment.

## Conclusion

This evaluation shows that API-based GenAI can effectively assist statistical programmers across SAS Studio, and Databricks environment. When combined with structured prompt engineering and governance controls, ChatGPT enhances productivity while maintaining reproducibility and auditability.

## References

1. Chen, M. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.
2. Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., ... & Xiong, C. (2022). Codegen: An open large language model for code with multi-turn program synthesis. arXiv preprint arXiv:2203.13474.
3. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... & McGrew, B. (2023). Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

## Acknowledgments

I greatly acknowledge the support and guidance from my supervisor Antonio Lovatin and DX-DSS-LT and GenAI Innovation Team.

## Contact Information

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

**Ajay Gupta (M.S.)** is Director of Statistical Programming at Daiichi Sankyo U.S., with over 20 years of experience in data standardization, technical leadership, and systems development across the CRO and

pharmaceutical sectors. He holds a master's degree in biomedical engineering from Louisiana Tech University and has been a regular SAS and Pharma SUG conference presenter since 2010, covering topics such as CDISC, RWD/RWE, and advanced analytics tools. He can be contacted at [ajay.gupta@daiichisankyo.com](mailto:ajay.gupta@daiichisankyo.com).

**Misikir Tesfaye, MSc, MPH**, is Manager in Data Engineering within statistical programming and Innovation at Daiichi Sankyo Inc currently. He has more than 20 years of experience in clinical, health care and pharmaceutical analytics. His expertise includes HEOR,HTA and real-world evidence analytics. He is an advanced user of SAS,R and python for statistical programming , data science, and machine learning applications. He can be contacted at [misikir.tesfaye@daiichisankyo.com](mailto:misikir.tesfaye@daiichisankyo.com).

## DISCLAIMER

The content of this paper is the works of the authors and does not necessarily represent the opinions, recommendations, or practices of Daiichi Sankyo.

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