This paper explores the potential benefits of incorporating ChatGPT, a state-of-the-art natural language processing model, in statistical programming within the pharmaceutical industry. By leveraging ChatGPT’s capabilities, this technology can save time, money, and most importantly, your sanity. Programming often leads to frustration, anxiety, and sleepless nights trying to solve complex problems. Various applications and techniques that harness the power of ChatGPT will be described to reduce all of these. In a world where artificial intelligence threatens to take our jobs, this paper suggests methods of tapping into the untapped potential of ChatGPT to empower programmers with innovative tools, thereby increasing their value. When programming issues arise, no longer will you need to worry about judgement or hostility from others on online forums, particularly the wrath experienced when not including a reproducible example. ChatGPT is a powerful tool we have yet to fully leverage, and its benefits extend well beyond our imaginations, let alone this paper.

DISCLAIMER
Let me begin by stating I do not pretend to know anything about how large language models like ChatGPT work. I simply interact with it and try to ask the best questions that will produce the results I am looking for. This is like how most programmers do not fully understand how code is compiled yet can still be an expert programmer. I also consider myself an advanced SAS programmer, and an intermediate R programmer. The techniques I describe in this paper can be used by any level of programmer that use SAS or R, but I highly recommend at least an intermediate understanding of the language with which you are working. ChatGPT is not perfect (but it is getting better). There is a warning on OpenAI’s ChatGPT website: “ChatGPT can make mistakes. Consider checking important information.” Finally, I mainly use OpenAI’s ChatGPT 3.5.

INTRODUCTION
ChatGPT is an exciting new technology that has the potential to change the world in ways we can barely imagine. I have heard many opinions on it, ranging from simply being a glorified search engine to being a phenomenal tool for generating wealth. Personally, I did not know what to make of it when it was first introduced. I initially tried using it to resolve disputes. For instance, it helped clarify a rare situation in a game of 8 ball pool between my brother and me where we did not know the rule to be applied. Another unusual situation happened during my son’s baseball game. Parents from opposing teams were arguing over a rare rule that was enforced. In both situations, ChatGPT was able to instantly provide clarification without having to sort through dozens of web pages using a search engine, or searching through a rule book that may not be available at that moment. I began to see the power in this new technology, and the potential it had to replace certain jobs, like customer service personnel, legal assistants, and even physicians. I wondered if it could replace the job of a statistical programmer. I sat down and started talking with ChatGPT. The first few programming-related questions I asked generated answers that made me believe that one day it could replace the job of a statistical programmer. However, the more I communicated with it, the more I realized it does not have to replace us. Instead, it could make us better programmers. This paper explores how to integrate ChatGPT into our daily work as statistical programmers by looking at situations where this technology can be used. Specific prompts I use that make it a powerful tool in my arsenal, along with practical examples, will be described to show how to begin harnessing its power.

EXPLORATION OF PROGRAMMING LANGUAGES
When I began talking with ChatGPT, I wanted to see what it knew about the programming languages we use. I sent it simple instructions to create a SAS program used to create a common output in the pharmaceutical industry. Here is the prompt:

You: Can you write me a SAS program that merges ADSL with ADVS by USUBJID, then calculate summary statistics (n, mean, std, min, med, max) by TRT01P for the variables AVAL and CHG, then run an MMRM analysis for the variable CHG using PROC MIXED with variables SEX as a categorical covariate and AGE as a continuous covariate and calculate LSMEANS?

ChatGPT: Certainly! Here's a SAS program that performs the tasks you mentioned:

The result generated in this case is shown in Appendix Output 1a.

We can also use a similar prompt to request the program be written in R. Here is the prompt:

You: Can you write me an R program that merges ADSL with ADVS by USUBJID, then calculate summary statistics (n, mean, std, min, med, max) by TRT01P for the variables AVAL and CHG, then run an MMRM analysis for the variable CHG using LMER with variables SEX as a categorical covariate and AGE as a continuous covariate and calculate LSMEANS?

ChatGPT: Sure, here's an R program that performs the requested tasks:

The result generated in this case is shown in Appendix Output 1b.

The resulting code is an impressive attempt at generating results for a common analysis. It may not be perfect, but it certainly is a good start we can begin working with.

It should be noted that in my experience, I believe ChatGPT is a better R programmer than a SAS programmer. For example, the SAS code it produces tends to be unreliable when it comes to sorting data. You may need to tell it how and when sorting needs to be done.

**CODE COMMENTING MADE EFFORTLESS**

One of my least favorite activities when programming is commenting code. Using a single prompt, ChatGPT can let me remove a substantial portion of this tedious task from my day-to-day work. Here is the prompt:

You: Can I send you a SAS program to comment?

ChatGPT: Of course! I'd be happy to help you with commenting your SAS program. Please go ahead and provide the SAS code you'd like me to review and comment on.

The resulting commented SAS code from this prompt is shown in Appendix Output 2a.

We can also use a similar prompt to request the same thing in R:

You: Can I send you an R program to comment?

ChatGPT: Of course! Feel free to paste your R program here, and I'll be happy to provide comments and suggestions.

The resulting commented R code from this prompt is shown in Appendix Output 2b.

These are simple examples of what ChatGPT can manage when commenting code. I have had particularly good success submitting long and complicated code. ChatGPT is able to add comments appropriately.
WHITESPACE ENHANCEMENT FOR READABILITY

Another example of ChatGPT’s capability that I use is improving whitespace in my programs. Whitespace is any space, tab, or carriage return used to separate information in the code. Proper use of whitespace helps with readability, particularly when your code is going to be read by another programmer or reviewer. Both SAS and R have their own functionalities for cleaning up whitespace: SAS Enterprise Guide employs Ctrl+Shift+B, while RStudio utilizes Ctrl+Shift+A. I use these functionalities routinely, but I now prefer how ChatGPT handles whitespace. In my opinion, it is a lot cleaner, better structured, and more condensed. Here is the prompt:

**You:** Can I send you a SAS program to improve the whitespace?

**ChatGPT:** Certainly! Feel free to share the SAS program, and I'll help improve the whitespace and formatting for better readability.

**You:**

data temp;set dsin.temp;where age>65;run;
proc sort data=temp;by usubjid;run;

**ChatGPT:** Sure, here's your SAS program with improved whitespace formatting for better readability:

data temp;
  set dsin.temp;
  where age > 65;
run;

proc sort data=temp;
  by usubjid;
run;

We can also use a similar prompt to request the same thing in R:

**You:** Can I send you an R program to improve the whitespace?

**ChatGPT:** Yes, absolutely! Feel free to share the R program with me, and I'll help you improve the whitespace for better readability.

**You:**

estimate <- dsin %>%
dplyr::filter(type == "LSMean") %>%
dplyr::select(param, time, trt, result)
stderr <- filtered.data %>%
dplyr::filter(type == "Std Err") %>%
dplyr::distinct(param, time, trt, result)
final <- dplyr::left_join(estimate, stderr, by = c("param", "time", "trt")) %>%
dplyr::rename(estimate = result.x, stderr = result.y) %>%
dplyr::mutate(stderr = ifelse(is.na(stderr), 0, stderr))

**ChatGPT:** Certainly! Here's your R program with improved whitespace formatting for better readability:

estimate <- dsin %>%
dplyr::filter(type == "LSMean") %>%
dplyr::select(param, time, trt, result)
stderr <- filtered.data %>%
dplyr::filter(type == "Std Err") %>%
dplyr::distinct(param, time, trt, result)
final <- dplyr::left_join(estimate, stderr, by = c("param", "time", "trt")) %>%
dplyr::rename(estimate = result.x, stderr = result.y) %>%
dplyr::mutate(stderr = ifelse(is.na(stderr), 0, stderr))
CROSS-LANGUAGE CODE CONVERSION

The first time I used ChatGPT for a work-related topic was when I needed to convert a SAS program into an equivalent version in the R programming language. I was not sure if it was even capable of doing this task. I thought I would just ask it if it were possible and see what happens. During our conversation, it not only confirmed that yes, it would be able to convert SAS code into R code, but it added that it would also be able to do the reverse. This was the first time I realized ChatGPT's potential. It made me realize it could eventually replace the job of a statistical programmer. Here is the prompt:

You: Can you help me convert SAS code into R code?

ChatGPT: Certainly! I'd be happy to help you convert SAS code into R code. Please provide the SAS code that you would like to convert, and I'll do my best to assist you.

You:

data keep_base_subs(keep=usubjid);
  set advs0(keep=usubjid abfl aval);
  where abfl="Y" and aval~=.;
run;

proc sort data=keep_base_subs nodupkey;
  by usubjid;
run;

data advs1;
  merge advs0(in=a) keep_base_subs(in=b);
  by usubjid;
  if a and b;
run;

data keep_post_subs(keep=usubjid);
  set advs1(keep=usubjid abfl chg);
  where abfl~="Y" and chg~=.;
run;

proc sort data=keep_post_subs nodupkey;
  by usubjid;
run;

data advs;
  merge advs1(in=a) keep_post_subs(in=b);
  by usubjid;
  if a and b;
run;

The result generated in this case is shown in Appendix Output 3a.

Here is another example of a prompt:

You: Can you help me convert SAS code into R code?

ChatGPT: Certainly! I'd be happy to help you convert SAS code into R code. Please provide the SAS code that you would like to convert, and I'll do my best to assist you.

You:
data _null_; set files;
call symput("num_files",compress(put(_n_,best12.)));  
call symput("filename"||compress(put(_n_,best12.)),strip(name));  
run;

ChatGPT: Certainly! The SAS code you provided seems to be reading data from a dataset called `files`, and then generating macro variables based on the data. Below is the equivalent R code using the `data.table` package to read data and generate variables:

```r
# Assuming 'files' is a data.table
library(data.table)

# Read your data into a data.table
files <- fread("your_file_path.csv")

# Create num_files macro variable
num_files <- nrow(files)
assign("num_files", num_files, envir = .GlobalEnv)

# Create filename1, filename2, ..., filen macro variables
for (i in 1:num_files) {
  filename_i <- files[i, name]
  assign(paste0("filename", i), filename_i, envir = .GlobalEnv)
}
```

We can also use a similar prompt to request R code be converted into SAS:

**You:** Can you help me convert R code into SAS code?

**ChatGPT:** Of course! Please provide the R code you'd like to convert, and I'll do my best to assist you in translating it into SAS code.

**You:**

```r
library(dplyr)
advs0 <- read.csv("advs0.csv")  # Assuming advs0 is a CSV file
keep_base_subs <- advs0 %>%
  filter(ablf == "Y" & !is.na(aval)) %>%
  select(usubjid)

keep_base_subs <- keep_base_subs[order(keep_base_subs$usubjid), ]
advs1 <- merge(advs0, keep_base_subs, by = "usubjid", all.x = TRUE, all.y = FALSE) %>%
  filter(lis.na(ablf) & lis.na(chg))

keep_post_subs <- advs1 %>%
  filter(ablf != "Y" & lis.na(chg)) %>%
  select(usubjid)

keep_post_subs <- keep_post_subs[order(keep_post_subs$usubjid), ]
advs <- merge(advs1, keep_post_subs, by = "usubjid", all.x = TRUE, all.y = FALSE)
advs <- advs[complete.cases(advs), ]
```

The result generated in this case is shown in Appendix Output 3b.
I have submitted long and complicated programs with success. Note that often we need to submit chunks of code at a time for ChatGPT to convert, then assemble all the chunks together afterwards. I don’t recommend sending hundreds of lines of code at a time and expecting a perfect response.

**EFFICIENCY BOOST THROUGH CODE OPTIMIZATION**

One of the most powerful uses of ChatGPT is to make your code more efficient. This is particularly useful when you have programs that have runtimes that are hours-long. There might be redundancies or inefficiencies that can be identified. For example, I created a piece of code that calculates summary statistics using two separate procedures (one for AVAL, one for CHG). Here is the prompt:

```
You: Can I send you a SAS program to optimize for efficiency?

ChatGPT: Certainly! I’d be happy to help you optimize your SAS program for efficiency. Please provide the details of your program or the specific parts you’re looking to optimize, and I’ll do my best to assist you.

You:
proc means data=merged_data n mean std min median max;
  by common_variable;
  var AVAL;
  output out=summary_stats(drop=_type_ _freq_)
    n=n_AVAL mean=mean_AVAL std=std_AVAL min=min_AVAL median=median_AVAL
    max=max_AVAL;
run;

proc means data=merged_data n mean std min median max;
  by common_variable;
  var CHG;
  output out=summary_stats(drop=_type_ _freq_)
    n=n_CHG mean=mean_CHG std=std_CHG min=min_CHG median=median_CHG
    max=max_CHG;
run;

ChatGPT: Certainly! You can combine these two PROC MEANS steps into one step by using the VAR statement with multiple variables. This will save you from scanning the dataset twice and thus make your code more efficient. Here’s the optimized version:

```
proc means data=merged_data n mean std min median max;
  by common_variable;
  var AVAL CHG;
  output out=summary_stats(drop=_type_ _freq_)
    n=n_AVAL mean=mean_AVAL std=std_AVAL min=min_AVAL median=median_AVAL
    max=max_AVAL
    n=n_CHG mean=mean_CHG std=std_CHG min=min_CHG median=median_CHG
    max=max_CHG;
run;
```

We can also use a similar prompt to request the same thing in R:

```
You: Can I send you an R program to optimize for efficiency?

ChatGPT: Of course! Please feel free to share your R program, and let me know what specific aspects you’d like to optimize for efficiency. Whether it’s improving runtime performance, reducing memory usage, or enhancing the overall code structure, I’ll do my best to provide suggestions for optimization.
```
You:
summary_stats_aval <- merged_data %>%
group_by(TRT01P) %>%
summarize(
  n = n(),
  mean_AVAL = mean(AVAL, na.rm = TRUE),
  std_AVAL = sd(AVAL, na.rm = TRUE),
  min_AVAL = min(AVAL, na.rm = TRUE),
  median_AVAL = median(AVAL, na.rm = TRUE),
  max_AVAL = max(AVAL, na.rm = TRUE)
)

summary_stats_chg <- merged_data %>%
group_by(TRT01P) %>%
summarize(
  mean_CHG = mean(CHG, na.rm = TRUE),
  std_CHG = sd(CHG, na.rm = TRUE),
  min_CHG = min(CHG, na.rm = TRUE),
  median_CHG = median(CHG, na.rm = TRUE),
  max_CHG = max(CHG, na.rm = TRUE)
)

ChatGPT: Certainly! Your code seems to be calculating summary statistics for two variables (AVAL and CHG) grouped by the variable TRT01P. Here are a few suggestions to optimize this code for efficiency:

1. Combine into a Single Summary Operation: Since you're summarizing the same data frame twice but for different variables, you can combine both operations into a single operation to avoid redundant group-by operations.

2. Parallel Processing: If your dataset is very large and your machine has multiple cores, you might benefit from parallel processing. However, in R, parallel processing can be a bit complex to implement due to limitations in parallelizing operations involving data frames. But it's still worth exploring if your dataset is large enough.

Here's the optimized version combining both summary operations:

```r
library(dplyr)

summary_stats <- merged_data %>%
group_by(TRT01P) %>%
summarize(
  n = n(),
  mean_AVAL = mean(AVAL, na.rm = TRUE),
  std_AVAL = sd(AVAL, na.rm = TRUE),
  min_AVAL = min(AVAL, na.rm = TRUE),
  median_AVAL = median(AVAL, na.rm = TRUE),
  max_AVAL = max(AVAL, na.rm = TRUE),
  mean_CHG = mean(CHG, na.rm = TRUE),
  std_CHG = sd(CHG, na.rm = TRUE),
  min_CHG = min(CHG, na.rm = TRUE),
  median_CHG = median(CHG, na.rm = TRUE),
  max_CHG = max(CHG, na.rm = TRUE)
)
```
ChatGPT includes an explanation on how it makes the code more efficient. Note that often we need to submit chunks of code at a time for ChatGPT to make more efficient, then assemble all the chunks together afterwards. I don’t recommend sending hundreds of lines of code at a time and expecting a perfect response. I might then submit the whole program and see if it can be made even more efficient. Note that the resulting program is not always perfect, so knowledge of both SAS and R are needed, and testing is highly recommended.

READABILITY ENHANCEMENT FOR BETTER UNDERSTANDING

Another powerful use of ChatGPT is to make your code more readable. This is particularly useful when your code is going to be read by another programmer or reviewer. There might be complicated sections of your code where ChatGPT has an approach that is clearer and easier to follow. Here is the prompt:

You: Can I send you SAS code to optimize for readability?

ChatGPT: Certainly! Feel free to share your SAS code, and I’ll do my best to help you optimize it for readability. Please provide the relevant portions of the code or specify any specific concerns you have about readability.

You:
%macro example(var=);
  %if &var.=stderrmu %then
    %let round_num=3;
  %else %if &var.=diff | &var.=lower | &var.=upper %then
    %let round_num=1;
  %else %let round_num=2;;
  %put &round_num.;
%mend example;

%example(var=stderrmu);

ChatGPT: Your provided SAS macro code is already quite concise, but if you want to enhance readability, you might consider the following adjustments:

%macro example(var=);
  %local round_num;
  %if &var. = stderrmu %then %do;
    %let round_num = 3;
  %end;
  %else %if &var. = diff | &var. = lower | &var. = upper %then %do;
    %let round_num = 1;
  %end;
  %else %do;
    %let round_num = 2;
  %end;
  %put &round_num.;
%mend example;

%example(var=stderrmu);

Changes made for readability:

1. Added %local round_num; to explicitly declare round_num as a local macro variable.
2. Used %do; and %end; to encapsulate the code within the %if and %else blocks. While this is not strictly necessary, it can improve code readability, especially as your code expands.  
3. Added indentation for better visual structure.  

These changes aim to make the code more organized and easier to follow. Feel free to adjust according to your preferences and team's coding standards.

We can also use a similar prompt to request the same thing in R:

**You:** Can I send you R code to optimize for readability?

**ChatGPT:** Of course! I'd be happy to help optimize your R code for readability. Please go ahead and share the code you'd like to improve, and I'll provide suggestions to make it more readable.

**You:**

```r
if (input_AVISIT == "ALL VISITS") plot.type <- "time course plot" else plot.type <- "bar plot"
if (input_AVISIT == "ALL VISITS") pp <- output_plot(plot.type = plot.type) else pp <- output_plot(plot.type = plot.type2)
```

**ChatGPT:** To improve readability, you can use consistent indentation and line breaks. Here's an optimized version of your R code:

```r
# Determine plot type based on input_AVISIT
plot.type <- if (input_AVISIT == "ALL VISITS") {  
  "time course plot"
} else {
  "bar plot"
}

# Determine the appropriate output_plot function based on plot type
pp <- output_plot(plot.type = plot.type)
```

In this version, I've used curly braces to create code blocks for better structure. This makes it clear which expressions are part of the conditional blocks and improves overall readability. Feel free to adjust it based on your coding style preferences.

ChatGPT includes an explanation on how it makes the code more readable.

**CODE DEBUGGING UNVEILED**

Debugging code is often a difficult and time-consuming process. ChatGPT can be used to discover hard-to-find syntax issues in both SAS and R. It also recognizes potential flaws in the structure of your program. Here is the prompt:

**You:** Can you help me debug my SAS code?

**ChatGPT:** Certainly! I'd be happy to help you debug your SAS code. Please provide the relevant portions of your code along with a description of the issue or error you're encountering. If possible, include any error messages or unexpected behavior you've observed. This will help me understand the problem and provide you with the best assistance.

**You:**

data temp2;
  set temp1;
  do i=1 to 10;
temp=i
end;
run;

ChatGPT: It looks like there might be a small syntax issue in your code. You're missing a semicolon after the assignment statement inside the `do` loop. Here's the corrected code:

data temp2;
  set temp1;
  do i = 1 to 10;
    temp = i; /* Added a semicolon here */
  end;
run;

This correction ensures that the assignment statement `temp=i` is properly terminated with a semicolon. If you encounter any specific errors or issues, feel free to provide more details, and I'll be happy to assist further.

We can also use a similar prompt to request the same thing in R:

You: Can you help me debug my R code?

ChatGPT: Certainly! Please provide the R code that you would like me to debug. Additionally, let me know if you have specific error messages, unexpected behavior, or any particular issues you'd like assistance with. I'll do my best to help you debug the code.

You:
plot.type <- "time course plot"
plot.type <- "bar plot"
pp <- output_plot(plot.type = ifelse(input_AVISIT == "ALL VISITS", plot.type, plot.type2))

ChatGPT: It seems like the issue lies in how you're assigning the `plot.type` variable. You're overwriting the value of `plot.type` immediately after setting it, which renders the first assignment useless. Here's a revised version of your code:

# Determine plot type based on input_AVISIT
plot.type <- ifelse(input_AVISIT == "ALL VISITS", "time course plot", "bar plot")

# Generate plot using the determined plot type
pp <- output_plot(plot.type = plot.type)

In this corrected version, the ifelse() function is used to conditionally assign the value of plot.type based on the value of input_AVISIT. Then, the determined plot.type is used as an argument in the output_plot() function call. This should resolve the issue with your code.

ChatGPT includes details on what issues it has found, and provides an explanation on how it has fixed them.

DYNAMIC CODING SOLUTIONS VIA CONVERSATIONS

One valuable practice I often use is to ask ChatGPT for help deriving coding solutions for various situations. I use a few sentences to describe what I would like it to create. I then find myself having conversations to narrow down exactly what I need from it. Here is an example of a prompt:
**You:** In SAS, I'm trying to concatenate multiple variables in a dataset, separated by a space, into a new variable. The variables all start with "var", and there are 100 of them.

**ChatGPT:** In SAS, you can use an array and a loop to concatenate multiple variables into a new variable. Since you have 100 variables named `var1` through `var100`, it's more efficient to use an array rather than writing out each variable individually.

Here's an example code to concatenate these variables separated by a space:

```sas
data new_dataset;
  set your_existing_dataset; /* Replace 'your_existing_dataset' with your actual dataset name */

  /* Create an array to hold the variables */
  array var_array{100} var1-var100;

  /* Initialize the new concatenated variable */
  new_var = "";

  /* Loop through the array and concatenate the values */
  do i = 1 to 100;
    new_var = catx(' ', new_var, var_array[i]);
  end;

  /* Optionally, drop the loop index variable if you don't need it in the output */
  drop i;

  /* Optionally, drop the original 'var' variables */
  drop var1-var100;
run;
```

Here's what the code does:

1. `array var_array{100} var1-var100;`: This line creates an array named `var_array` with 100 elements, corresponding to your variables `var1` through `var100`.
2. `new_var = "";`: This initializes the new variable `new_var` to an empty string.
3. `do i = 1 to 100;`: This starts a loop that iterates through the array elements.
4. `new_var = catx(' ', new_var, var_array[i]);`: This line concatenates the values of `var_array[i]` (i.e., `var1`, `var2`, ..., `var100`) to `new_var`, separated by a space.
5. `drop i;`: If you don't need the loop index variable `i` in the output dataset, you can drop it.
6. Optionally, you can add `drop var1-var100;` to remove the original `var` variables if you no longer need them in your final dataset.

Remember to replace `your_existing_dataset` with the actual name of your dataset in the `set` statement.

We can also create coding solutions in the R language. Here is an example of a prompt:

**You:** I have an R function called "pval_symbol" that has 2 inputs called "pvalue" and "symbol". I would like to use `lapply` to call this function, but I am having a problem because the function "pval_symbol" has 2 arguments. can you help me?
ChatGPT: Yes, you can use `lapply` with an anonymous function to apply your function `pval_symbol` to a list of inputs. Here's how you can do it:

```r
# Define your function
pval_symbol <- function(pvalue, symbol) {
    # Your function logic here
    # For example:
    paste("Symbol:" , symbol, "P-value:" , pvalue)
}

# Create a list of inputs
pvalue_list <- c(0.05, 0.01, 0.001)
symbol_list <- c("A", "B", "C")
input_list <- list(pvalue = pvalue_list, symbol = symbol_list)

# Use `lapply` to apply the function to each element of the list
result <- lapply(seq_along(input_list[[1]]), function(i) {
    pval_symbol(input_list$pvalue[[i]], input_list$symbol[[i]])
})

# Output the result
print(result)
```

I was curious if this could also be done another way:

You: Can this be done another way?

ChatGPT: Yes, you can achieve the same result using `mapply`. `mapply` is designed for exactly this situation where you want to apply a function to multiple arguments in parallel.

Here's how you can use `mapply`:

```r
# Define your function
pval_symbol <- function(pvalue, symbol) {
    # Your function logic here
    # For example:
    paste("Symbol:" , symbol, "P-value:" , pvalue)
}

# Create a list of inputs
pvalue_list <- c(0.05, 0.01, 0.001)
symbol_list <- c("A", "B", "C")

# Use `mapply` to apply the function to each pair of elements
result <- mapply(pval_symbol, pvalue_list, symbol_list)

# Output the result
print(result)
```

At the time of this exchange, I was not aware of the function `mapply`. This ended up being a learning opportunity.

Here is another example of a prompt:
You: I would like to create R code. I have 3 datasets named data1, data2, data3. For each dataset, I would like to first check if the variable "avisit" exists. If it does, then I would like to create a complete list of this variable in all of the datasets.

ChatGPT: Sure, you can achieve this in R using a loop to iterate over your datasets and check if the variable "avisit" exists in each dataset. If it does, you can create a list of all the values of "avisit" across all datasets. Here's how you can do it:

```r
# List of your datasets
datasets <- list(data1, data2, data3)

# Initialize an empty list to store values of "avisit"
avisit_values <- list()

# Loop through each dataset
for (i in seq_along(datasets)) {
  # Check if "avisit" exists in the current dataset
  if ("avisit" %in% names(datasets[[i]])) {
    # If "avisit" exists, add its values to the list
    avisit_values[[i]] <- datasets[[i]]$avisit
  }
}

# Combine all values of "avisit" into a single vector
complete_avisit_list <- unlist(avisit_values)

# Print the complete list of "avisit" values
print(complete_avisit_list)
```

This code will iterate over each dataset ("data1", "data2", "data3"), check if the variable "avisit" exists in each dataset, and if it does, it will add its values to a list. Finally, it will combine all the values into a single vector called `complete_avisit_list`.

Sometimes ChatGPT will produce code that is not very efficient, or may make small mistakes, or misunderstand your request. It helps to know the programming language well enough that you can interact with it to narrow down exactly what code you are needing. In the example above, I noticed that creating a list of datasets initially might be resource-intensive, given they are large datasets. Because of this, I asked ChatGPT if it can do the same thing without initially creating a list of the datasets:

You: Is there a way of doing this without initially creating a list of the datasets?

ChatGPT: Yes, you can achieve this without creating a list of datasets explicitly. You can use `mget()` function to get the datasets by their names and then iterate over them. Here's how you can do it:

```r
# Get the names of your datasets
dataset_names <- c("data1", "data2", "data3")

# Initialize an empty list to store values of "avisit"
avisit_values <- list()

# Loop through each dataset name
for (name in dataset_names) {
  # Check if "avisit" exists in the current dataset
  if ("avisit" %in% names(get(name))) {
    # If "avisit" exists, add its values to the list
  }
}
avisit_values[[name]] <- get(name)$avisit
}

# Combine all values of "avisit" into a single vector
complete_avisit_list <- unlist(avisit_values)

# Print the complete list of "avisit" values
print(complete_avisit_list)

This code will iterate over each dataset name (`data1`, `data2`, `data3`), use `get()` to retrieve the dataset by its name, check if the variable "avisit" exists in each dataset, and if it does, it will add its values to a list. Finally, it will combine all the values into a single vector called `complete_avisit_list`.

ChatGPT is always happy to accommodate any requests you may have. If you ever see that it made a mistake, then tell it that it made a mistake. It will apologize, then make another attempt at the solution.

### DATASET CREATION ENDEAVORS

One of the biggest endeavors I have been pursuing is getting ChatGPT to generate code used to create datasets. Whether they are SDTM, ADaM, or custom datasets, I would like to simply send a set of specifications via a prompt and have ChatGPT write the code for me. The results produced are very promising but are far from perfect. I believe we could be to the point very soon where ChatGPT will produce flawless code, but for now the code produced should be considered a good start, rather than a final product. Here is the prompt used to create the DM domain with several variables:

**You:** Generate a SAS script to create a dataset named `dm` using SAS datasets (`civ1001`, `dm1001`, `ds2001`, `ex1001`) located in `/study_location/final/data/raw`. Here are the specifications:

- Add a new column `STUDYID`, which is the variable `STUDYID` in the `dm1001` dataset.
- Add a new column `DOMAIN`, which is the constant value "DM".
- Add a new column `USUBJID`, which is the concatenation of variables `STUDYID`, `SITE`, and `SUBJID` in the `dm1001` dataset, separated by '-'. Leading 0's are added to make 5 digits for variables `SITE` and `SUBJID`. `SITE` and `SUBJID` are character variables.
- Add a new column `SUBJID`, which is the variable `SUBJID` in the `dm1001` dataset.
- Add a new column `SITEID`, which is the variable `SITE` in the `dm1001` dataset.
- Add a new column `ETHNIC`, which is the variable `ETHNIC` in the `dm1001` dataset. If ETHNIC= "NOT APPLICABLE", set to "NOT REPORTED".
- Add a new column `SEX`, which is the variable `SEX` in the `dm1001` dataset.
- Add a new column `RACE`, which is the concatenation of variables `RACE_1`, `RACE_2`, `RACE_3`, `RACE_4`, or `RACE_5` from the `dm1001` input dataset is not blank. Otherwise, set to the concatenation of variables `RACE_1`, `RACE_2`, `RACE_3`, `RACE_4`, and `RACE_5`. `RACE_1`, `RACE_2`, `RACE_3`, `RACE_4`, and `RACE_5` are character columns.
- Add a new column `RFICDTC`, which is the variable `DSSTDAT_IC` from `ds2001` dataset, keeping only variables `SUBJID`, and `DSSTDAT_IC`. Keep the minimum value this new variable for each `SUBJID`. Merge by `SUBJID`.
- Add a new column `RFXSTDTC`, which is the concatenation of variable `EXSTDAT`, "T", and variable `EXSTTIM` from `ex1001` dataset where EXOCCUR = "Y", keeping only variables `SUBJID`, `EXSTDAT`, `EXSTTIM`, and `EXOCCUR`. Keep the minimum value of this new variable for each `SUBJID`. Merge by `SUBJID`.

Exclude participants where the value of `SUBJID` matches the value of `PRVSUBJSFID` from the `civ1001` dataset, where the variable `PRVSUBJSFFLG` is set to "Y". Note that `PRVSUBJSFID` is a numeric variable and needs to be converted to character.

Keep only variables `STUDYID`, `DOMAIN`, `USUBJID`, `SUBJID`, `SITEID`, `ETHNIC`, `SEX`, `RACE`, `RFICDTC`, and `RFXSTDTC`.

Sort the `dm` dataset by `STUDYID`, `USUBJID`, and `SUBJID`.

Save the modified dataset as a CSV file in the specified location: `/study_location/final/output`.
The result is shown in Appendix Output 4a.

We can also use a similar prompt to request the same thing in R. The result generated in this case is shown in Appendix Output 4b.

**LEARNING AID FOR LANGUAGE PROFICIENCY**

For those who are strong programmers in one language (either R or SAS), but not both, I have found ChatGPT to be an invaluable resource for learning. When I started using ChatGPT to convert SAS programs into R programs, I would do this by sending small sections of code at a time, then seeing how ChatGPT converts them into R. This allowed me to see the translation of code I was familiar with into code that was new to me. Not only this, but it also occasionally produced code I was not expecting. This allowed me to see various ways of creating something that I might be able to implement elsewhere in the future. In addition, simply talking with ChatGPT about anything programming-related will help programmers of all skill levels. Recently, I have started using JavaScript to display results. I am not a JavaScript programmer, but ChatGPT was even helpful there.

When looking at code in an unfamiliar language, you can also submit the code to ChatGPT and ask it in plain language what the code is doing. Here is an example prompt:

**You:** Can I send you SAS code and have you explain in plain language what it is doing?

**ChatGPT:** Of course! Feel free to send me the SAS code, and I'll do my best to explain it to you in plain language.

**You:**

```
proc sql noprint;
  select distinct name into: tfl_id_column
    from dictionary.columns
    where libname = "WORK"
      and memname = "XLOA"
      and (upcase(name) like '%TFL%' and upcase(name) like '%ID%');
  select distinct name into: output_name_column
    from dictionary.columns
    where libname = "WORK"
      and memname = "XLOA"
      and (upcase(name) like '%OUTPUT%' and upcase(name) like '%NAME%');
quit;
```

**ChatGPT:** This SAS code is using PROC SQL to perform two separate queries on the dictionary.columns table. Let's break it down step by step:

(Steps omitted)

In summary, this SAS code is extracting column names from the "WORK.XLOA" dataset where the column names contain 'TFL' and 'ID' for tfl_id_column and 'OUTPUT' and 'NAME' for output_name_column. These column names are stored in macro variables for further use in the SAS code.

For those looking to improve their programming skills, you can use ChatGPT to create a course for you. It can be as specific as you would like. One useful course is for SAS programmers who would like to transition their skills into the R language. Here is an example prompt:
Sure! Transitioning from SAS to R can be a rewarding endeavor, as R offers powerful statistical and graphical capabilities along with a vibrant open-source community. Here's a beginner's course outline to help a SAS programmer make the switch:

Course Title: Transitioning from SAS to R: A Beginner's Guide

Course Overview:
This course is designed for SAS programmers who want to transition to R for data analysis, visualization, and statistical modeling. It covers the basic concepts of R programming, data manipulation, visualization, and statistical analysis commonly used in SAS.

Prerequisites:
- Basic understanding of programming concepts
- Familiarity with SAS programming language

Course Outline:
1. Introduction to R
   - Overview of R and its advantages over SAS
   - Installing R and RStudio
   - Introduction to RStudio interface
2. Basic R Syntax
   - Data types: vectors, matrices, lists, data frames
   - Variable assignment and basic arithmetic operations
   - Functions and basic control structures (if-else statements, loops)

Now that ChatGPT has created a skeleton of a course, you can ask it to expand on each module and provide examples and exercises for the student. Here is an example prompt:

You: Can you elaborate on the first module?

ChatGPT: Absolutely! The first module, "Introduction to R," serves as the foundation for the course, providing participants with an understanding of R and its advantages over SAS, as well as practical skills in setting up their R environment and navigating the RStudio interface.

Module 1: Introduction to R
1. Overview of R
   - Comparison between R and SAS in terms of features, flexibility, cost, and community support.
   - Introduction to the vast ecosystem of R packages for various statistical analyses, data visualization, and machine learning.
2. Installing R and RStudio
   - Step-by-step instructions on how to download and install R on different operating systems (Windows, macOS, Linux).
   - Installing RStudio, a powerful integrated development environment (IDE) for R, and its benefits in coding and project management.

You can keep asking for further elaborations until you feel comfortable with the material. You can also create courses that are specific to what you need. For example, you could create a course to learn Shiny or Rmarkdown.
**INFORMATION LIMITS**

I wanted to warn that ChatGPT can only handle a certain amount of information sent to it in one prompt. You may encounter the message “The message you submitted was too long, please reload the conversation and submit something shorter.” Here is the conversation I had with ChatGPT about this:

<table>
<thead>
<tr>
<th>You: What is the token limit when asking you a question?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ChatGPT</strong>: The token limit for a single interaction with the GPT-3.5 model, like asking a question, is 4096 tokens. This includes both input and output tokens. If your input is close to or exceeds this limit, you may need to truncate, omit, or otherwise reduce the text to fit within the constraint. Keep in mind that very long conversations may receive incomplete replies if they approach or exceed the token limit.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>You: How many letters are in 1 token?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ChatGPT</strong>: In the context of GPT-3.5, a token is not necessarily equivalent to a single letter. A token can be as short as one character or as long as one word. It's a unit of text that the model processes at once, and its length can vary based on the complexity of the language. For example, the word “apple” is one token, and the space character between two words is also considered one token. Therefore, the number of letters in a token depends on the specific characters present in that token. It's not a fixed correspondence of one letter per token.</td>
</tr>
</tbody>
</table>

**CONCLUSION**

This exploration into the integration of ChatGPT in statistical programming within the pharmaceutical industry shows an enormous potential, where the limit is your imagination. From generating code in multiple languages to optimizing efficiency, enhancing readability, and aiding in dynamic problem-solving, ChatGPT emerges as a versatile and invaluable tool for programmers. The good news is that ChatGPT does not need to replace our jobs as statistical programmers. Instead, it has the capacity to empower and enhance the capabilities of programmers. The invention of the calculator did not put statisticians out of business. You cannot give a calculator to anyone and tell them to do statistics. You still need expertise in the subject. ChatGPT has made its way into my daily use. It has the role of a problem solver and has the potential to launch an innovation era like we have never seen before in our industry. The key is to be specific with your prompts, but not too specific. Be broad with your prompts, but not too broad. There is a sweet spot. When you have a problem, just start having a conversation with ChatGPT to try and solve it. You can always ask follow-up questions to refine your answer. It is important to realize the limitations, like ChatGPT’s occasional imperfections, and the need for a good understanding of the language you are working with. Note that online forums do not encourage solutions to questions generated using artificial intelligence sources. Always avoid sending confidential information when conversing with ChatGPT. Finally, my most used prompt right now is done after writing a new program: “Can I submit a new R program to optimize for efficiency, improve whitespace, and add comments?” With this new and powerful resource at our fingertips, let us become even more valuable statistical programmers than we ever were!

**ACKNOWLEDGMENTS**

The author acknowledges the invaluable insights gained through interactions with ChatGPT and the support of the OpenAI community.

**CONTACT INFORMATION**

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

- Ian Sturdy
- Eli Lilly and Company
- sturdy_ian_macnicol@lilly.com

**APPENDIX**
/* Step 1: Merge ADSL with ADVS by USUBJID */
data merged_data;
merge ADSL (in=a) ADVS (in=b);
by USUBJID;
if a and b;
run;

/* Step 2: Calculate summary statistics for AVAL and CHG by TRT01P */
proc means data=merged_data n mean std min median max;
class TRT01P;
var AVAL CHG;
output out=summary_stats;
run;

/* Step 3: Run MMRM analysis for CHG using PROC MIXED */
proc mixed data=merged_data;
class TRT01P SEX;
model CHG = TRT01P SEX AGE / solution;
lsmeans TRT01P / cl;
run;

Output 1a. Output of SAS program created using ChatGPT.

# Load necessary libraries
library(dplyr)
library(lme4)
library(lsmeans)

# Assume ADSL and ADVS are data frames containing the respective data
# Merge ADSL with ADVS by USUBJID
merged_data <- merge(ADSL, ADVS, by = "USUBJID")

# Calculate summary statistics by TRT01P for AVAL and CHG
summary_stats <- merged_data %>%
group_by(TRT01P) %>%
summarize(n = n(),
          mean_AVAL = mean(AVAL, na.rm = TRUE),
          std_AVAL = sd(AVAL, na.rm = TRUE),
          min_AVAL = min(AVAL, na.rm = TRUE),
          median_AVAL = median(AVAL, na.rm = TRUE),
          max_AVAL = max(AVAL, na.rm = TRUE),
          mean_CHG = mean(CHG, na.rm = TRUE),
          std_CHG = sd(CHG, na.rm = TRUE),
          min_CHG = min(CHG, na.rm = TRUE),
          median_CHG = median(CHG, na.rm = TRUE),
          max_CHG = max(CHG, na.rm = TRUE))

# Run MMRM analysis for the variable CHG using LMER
mmrm_model <- lmer(CHG ~ TRT01P + SEX + AGE + (1|USUBJID), data = merged_data)

# Calculate LSMEANS
lsmeans_result <- lsmeans(mmrm_model, specs = c("TRT01P"), type = "response")

# Print summary statistics and LSMEANS
print(summary_stats)
print(lsmeans_result)

Output 1b. Output of R program created using ChatGPT.

/* Step 1: Frequency Analysis for Big Sample */
proc freq data=xxxx noprint;
  by avisitn avisit; /* Grouping data by avisitn and avisit variables */
  table trt01pn*trt01p / out=big_n; /* Cross-tabulation of trt01pn and trt01p, outputting results to big_n dataset */
quit;

/* Step 2: Frequency Analysis for Small Sample */
proc freq data=xxxx noprint;
  by avisitn trt01pn; /* Grouping data by avisitn and trt01pn variables */
  table incd / out=small_n(where=(incd=1)); /* Frequency of incd=1, outputting results to small_n dataset */
quit;

/* Step 3: Summary Statistics */
proc means data=xxxx noprint;
  by avisitn trt01pn; /* Grouping data by avisitn and trt01pn variables */
  var epis adurn; /* Variables to calculate means */
  output out=nepis sum=sum_epis sum_adurn; /* Outputting summary statistics to nepis dataset */
run;

/* Step 4: Merging Datasets */
data anl_ar1_results; /* Creating a new dataset */
  merge nsubs_big nsubs_small nepis; /* Merging datasets nsubs_big, nsubs_small, and nepis */
  by avisitn trt01pn; /* Merging by avisitn and trt01pn variables */
run;

/* Step 5: Sorting Data */
proc sort data=xxxx;
  by avisitn incd descending trt01pn usubjid; /* Sorting data by avisitn, incd (descending), trt01pn, and usubjid variables */
run;

/* Step 6: Logistic Regression */
ods listing close; /* Turning off listing output */
ods output convergencestatus=convergence_base; /* Outputting convergence status to convergence_base dataset */
proc logistic data=xxxx descending order=data; /* Performing logistic regression */
  by avisitn; /* Grouping data by avisitn variable */
  class trt01pn(descending) / param=glm; /* Specifying trt01pn as a class variable */
  model incd=trt01pn / firth link=logit maxiter=100; /* Model specification */
  lsmeans trt01pn / cl diff oddsratio ilink om; /* Estimating least squares means and confidence intervals */
  ods output diffs=diffs; /* Outputting differences to xdiffs dataset */
run;
ods listing; /* Turning on listing output */

Output 2a. Output of commented SAS program created using ChatGPT.
# Define a function for data validation
validation_function <- function(data, list_of_params, forest) {

    # Make a copy of the input data
    ir_data <- data

    # Create an empty list to store intermediate dataframes
    hold_ir_data <- list()

    # Check if there are parameters to filter the data
    if (length(list_of_params) != 0) {

        # Iterate over rows of the parameter matrix
        for (i in 1:nrow(list_of_params)) {

            # Create a list of conditions for each parameter set
            conditions <- lapply(seq_along(list_of_params[, ,]), function(j) {
                rlang::quo (!!as.symbol(names(list_of_params)[j]) == list_of_params[, j])
            })

            # Filter the data based on conditions
            ir_data <- dplyr::filter(ir_data, !!conditions)

            # Store the filtered data in the list
            hold_ir_data[[i]] <- ir_data
        }

        # Combine the filtered dataframes into a single dataframe
        final_data <- do.call("rbind", hold_ir_data)
    } else {

        # If no parameters provided, use the original data
        final_data <- ir_data
    }

    # Return the final filtered data
    return(final_data)

}

Output 2b. Output of commented R program created using ChatGPT.

# Load necessary libraries
library(dplyr)
library(readr)

# Read SAS dataset into R
adv0 <- read_sas("path/to/your/sas/file/adv0.sas7bdat")

# Extract necessary columns and filter rows
keep_base_subs <- adv0 %>%
    select(usubjid, ablfl, aval) %>%
    filter(ablfl == "Y", is.na(aval))
# Remove duplicates and sort by usubjid
keep_base_subs <- keep_base_subs %>%
distinct(usubjid) %>%
arrange(usubjid)

# Merge datasets and filter rows
advs1 <- merge(advs0, keep_base_subs, by = "usubjid") %>%
filter(!is.na(ablfl) & !is.na(aval))

# Extract necessary columns and filter rows
keep_post_subs <- advs1 %>%
select(usubjid, ablfl, chg) %>%
filter(ablfl != "Y", !is.na(chg))

# Remove duplicates and sort by usubjid
keep_post_subs <- keep_post_subs %>%
distinct(usubjid) %>%
arrange(usubjid)

# Merge datasets and filter rows
advs <- merge(advs1, keep_post_subs, by = "usubjid") %>%
filter(!is.na(ablfl) & !is.na(chg))

# Resulting dataset 'advs' now contains the desired data

Output 3a. Output of converting SAS program into R using ChatGPT.

/* Read in the CSV file */
proc import datafile='advs0.csv'
  out=advs0
dbms=csv;
run;

/* Keep observations where ablfl is "Y" and aval is not missing */
data keep_base_subs;
  set advs0;
  where ablfl = 'Y' and not missing(aval);
  keep usubjid;
run;

proc sort data=keep_base_subs;
  by usubjid;
run;

/* Merge advs0 with keep_base_subs */
data advs1;
  merge advs0(in=a) keep_base_subs(in=b);
  by usubjid;
  if a and b;
run;

/* Keep observations where ablfl is not "Y" and chg is not missing */
data keep_post_subs;
  set advs1;
where ablfl ne 'Y' and not missing(chg);
   keep usubjid;
run;

proc sort data=keep_post_subs;
   by usubjid;
run;

/*/ Merge advs1 with keep_post_subs */
data advs;
   merge advs1(in=a) keep_post_subs(in=b);
       by usubjid;
       if a and b;
run;

Output 3b. Output of converting R program into SAS using ChatGPT.

/*/ Set the library path */
libname rawdata '/study_location/final/data/raw';
libname output '/study_location/final/output';

/*/ Import datasets */
data civ1001;
   set rawdata.civ1001;
runk;

data dm1001;
   set rawdata.dm1001;
runk;

data ds2001;
   set rawdata.ds2001;
runk;

data ex1001;
   set rawdata.ex1001;
runk;

/*/ Convert PRVSUBJSFID to character */
data civ1001;
   set civ1001;
       PRVSUBJSFID_char = put(PRVSUBJSFID, 5.);
runk;

/*/ Exclude participants with PRVSUBJSFLG = "Y" */
data dm_excluded;
   merge dm1001 civ1001;
       by SUBJID;
       if PRVSUBJSFLG ne "Y";
runk;

/*/ Add new columns to the dm dataset */
data dm;
merge dm_excluded (keep = STUDYID SITE SUBJID ETHNIC SEX RACE_1 RACE_2 RACE_3 RACE_4 RACE_5) 
ds2001 (keep = SUBJID DSSTDAT_IC) 
ex1001 (where = (EXOCCUR = 'Y') keep = SUBJID EXSTDAT EXSTTIM EXOCCUR);

/* STUDYID, DOMAIN, USUBJID, SUBJID, SITEID, ETHNIC, SEX, RACE */
STUDYID = dm1001.STUDYID;
DOMAIN = 'DM';
USUBJID = catx('-', STUDYID, put(SITE, z5.), put(SUBJID, z5.));
SUBJID = dm1001.SUBJID;
SITEID = dm1001.SITE;
ETHNIC = ifc(ETHNIC = 'NOT APPLICABLE', 'NOT REPORTED', ETHNIC);
SEX = dm1001.SEX;

/* RACE */
if nmiss(RACE_1, RACE_2, RACE_3, RACE_4, RACE_5) > 1 then
  RACE = 'MULTIPLE';
else
  RACE = catx(' ', of RACE_1 RACE_2 RACE_3 RACE_4 RACE_5);

/* RFICDTC */
by SUBJID;
if first.SUBJID then RFICDTC = DSSTDAT_IC;

/* RFXSTDTC */
if EXOCCUR = 'Y' then do;
  EXDATETIME = catx('T', EXSTDAT, EXSTTIM);
  if first.SUBJID or EXDATETIME < RFXSTDTC then RFXSTDTC = EXDATETIME;
end;

keep STUDYID DOMAIN USUBJID SUBJID SITEID ETHNIC SEX RACE RFICDTC RFXSTDTC;
run;

/* Sort and save the dataset as CSV */
proc sort data=dm;
  by STUDYID USUBJID SUBJID;
run;

proc export data=dm
  outfile='/study_location/final/output/dm.csv'
  dbms=csv replace;
run;

Output 4a. Output of SDTM DM domain SAS program created using ChatGPT.

# Load required libraries
library(dplyr)
library(tidyr)

# Set the path to SAS datasets
sas_path <- '/study_location/final/data/raw'

# Import SAS datasets
civ1001 <- read_sas(file.path(sas_path, 'civ1001.sas7bdat'))
dm1001 <- read_sas(file.path(sas_path, 'dm1001.sas7bdat'))
ds2001 <- read_sas(file.path(sas_path, 'ds2001.sas7bdat'))
ex1001 <- read_sas(file.path(sas_path, 'ex1001.sas7bdat'))

# Convert PRVSUBJSFID to character in civ1001
civ1001$PRVSUBJSFID_char <- as.character(civ1001$PRVSUBJSFID)

# Exclude participants with PRVSUBJSFFLG = "Y"
dm_excluded <- dm1001 %>%
  left_join(civ1001 %>% filter(PRVSUBJSFFLG == "Y"), by = "SUBJID") %>%
  filter(is.na(PRVSUBJSFFLG))

# Add new columns to the dm dataset
dm <- dm_excluded %>%
  left_join(ds2001 %>% group_by(SUBJID) %>% summarize(RFICDTC = min(DSSTDAT_IC)), by = "SUBJID") %>%
  left_join(ex1001 %>% filter(EXOCCUR == "Y") %>% group_by(SUBJID) %>%
    summarize(RFXSTDTC = min(paste(EXSTDAT, "T", EXSTTIM))), by = "SUBJID") %>%
  mutate(
    STUDYID = dm1001$STUDYID,
    DOMAIN = "DM",
    USUBJID = paste(STUDYID, sprintf("%05d", SITE), sprintf("%05d", SUBJID), sep = "-"),
    SITEID = dm1001$SITE,
    ETHNIC = ifelse(ETHNIC == "NOT APPLICABLE", "NOT REPORTED", ETHNIC),
    RACE = ifelse(rowSums(!is.na(select(dm1001, RACE_1, RACE_2, RACE_3, RACE_4, RACE_5))) > 1, "MULTIPLE",
      paste(select(dm1001, RACE_1, RACE_2, RACE_3, RACE_4, RACE_5), collapse = " "))),
    EXOCCUR = ex1001$EXOCCUR
  ) %>%
  select(STUDYID, DOMAIN, USUBJID, SUBJID, SITEID, ETHNIC, SEX, RACE, RFICDTC, RFXSTDTC)

# Sort the dm dataset
dm <- dm %>% arrange(STUDYID, USUBJID, SUBJID)

# Save the modified dataset as a CSV file
output_path <- '/study_location/final/output'
write.csv(dm, file.path(output_path, 'dm.csv'), row.names = FALSE)

Output 4b. Output of SDTM DM domain R program created using ChatGPT.