PharmaSUG 2026 - Paper HT-451

Accelerating Code Translation and Generation Using ChatGPT®

David J. Bosak, Archytas Clinical Solutions
Brian Varney, Experis

ABSTRACT

Recent years have seen an explosion of use cases for generative AI. People are using AI to do research, write papers, summarize articles, and write code. At first, the quality of the output was not very good. Yet with each new release, the output is getting better and better. It is now feasible to perform tasks that previously would not be considered. In this paper, we will explore the topic of code translation and generation involving SAS® & R using ChatGPT®. The paper will first provide an overview of how to perform such a translation. We will then elaborate on a list of tips and tricks to make the machine translation better. Finally, we will point out some limitations of the technology and tell you what to watch out for. The paper will hopefully present you with enough information to try translating some code yourself.

INTRODUCTION

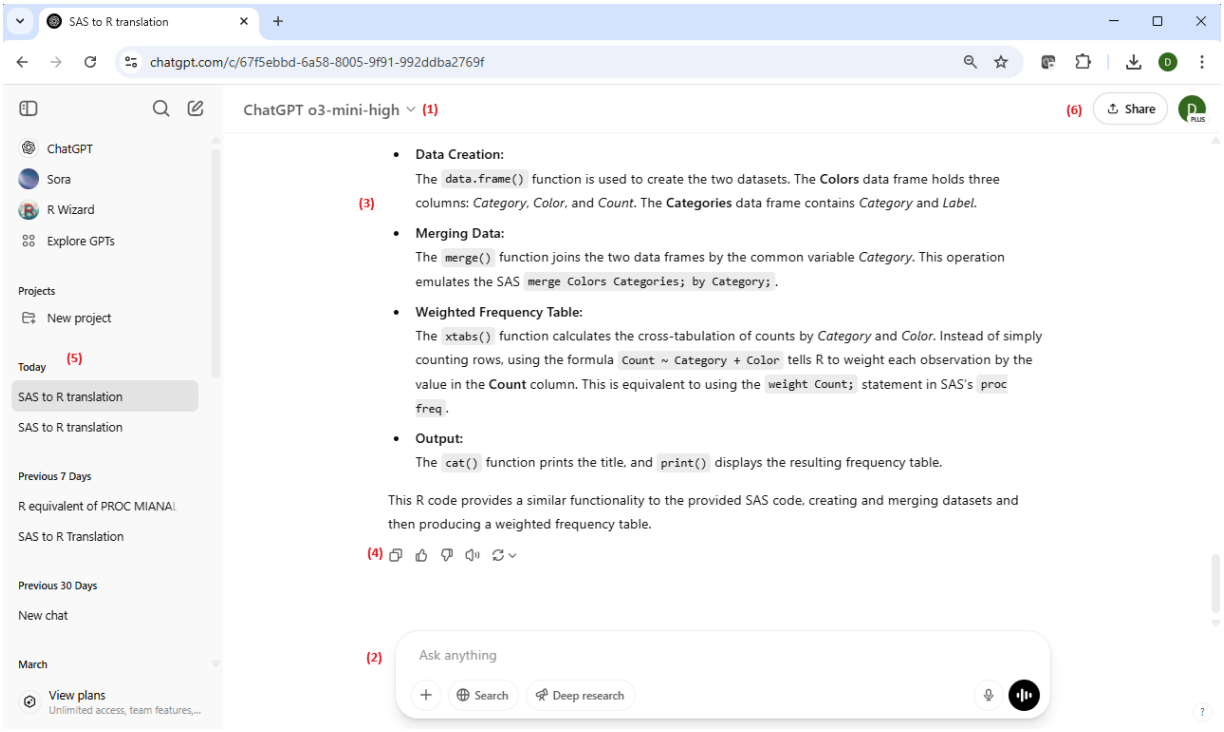
This paper intends to explore using generative AI (GenAI) to translate SAS programs directly to R programs. We will be using SAS 9.4 and R version 4.4.2. On the GenAI side we will be using ChatGPT's "Thinking 5.5" model.

The "Thinking 5.5" model is currently the most powerful free version from ChatGPT. This model is a "reasoning model" rather than a "predictive model". The reasoning models are far superior to predictive models. These models do much better with coding and logic tasks, have fewer hallucinations, and can "figure things out" themselves rather than regurgitate code published on the internet.

ChatGPT and R versions are changing frequently. At the time of this presentation, there may be newer versions available.

CHATGPT INTERFACE

Before starting to translate, let's first take a quick tour of the ChatGPT Interface. Here is an annotated screen shot of the main screen:



The annotations describe the most important items on the interface. They are as follows:

(1) The model version selector. This drop-down lets you chose the model you wish to use. For translation tasks, pick the highest version you have available.

(2) This input box is where you enter your prompt and code. You will then submit your prompt by pressing "enter" on the keyboard.

(3) This area is where the AI results are displayed. You will copy the generated code out of this section. There is a "copy" button on the code pane to make it easy for you.

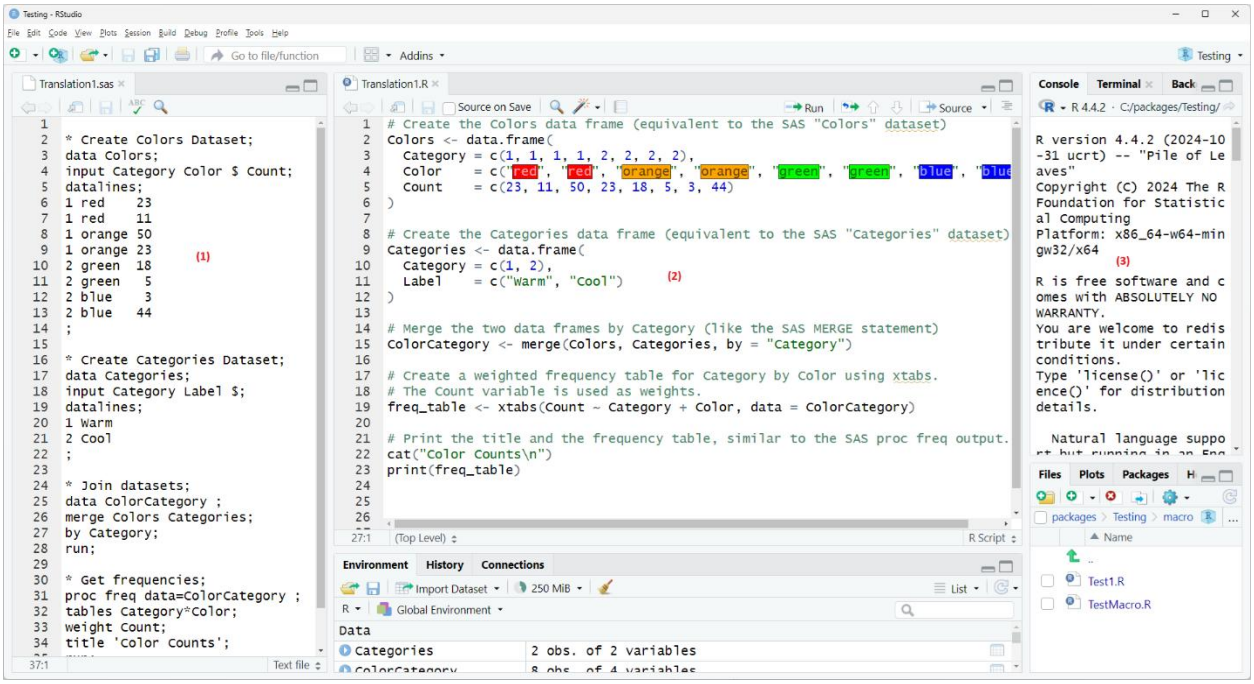
(4) A menu where you can provide feedback to the model.

(5) The left pane holds your chats. You can create a new conversation by clicking the "New Chat" icon at the top. Your old chats are saved for future reference, or for continuing a conversation.

(6) If you want to share a chat with a colleague, this button will allow you to do it.

RSTUDIO INTERFACE

For code translation from SAS to R, it is also helpful to rearrange your RStudio panels. The annotated screen shot below shows a three-panel layout. This layout allows you to look at your SAS code and your R code at the same time:



The sections of this layout are as follows:

(1) The SAS code you are translating from. Since a SAS program is a text file, you can just open it in RStudio.

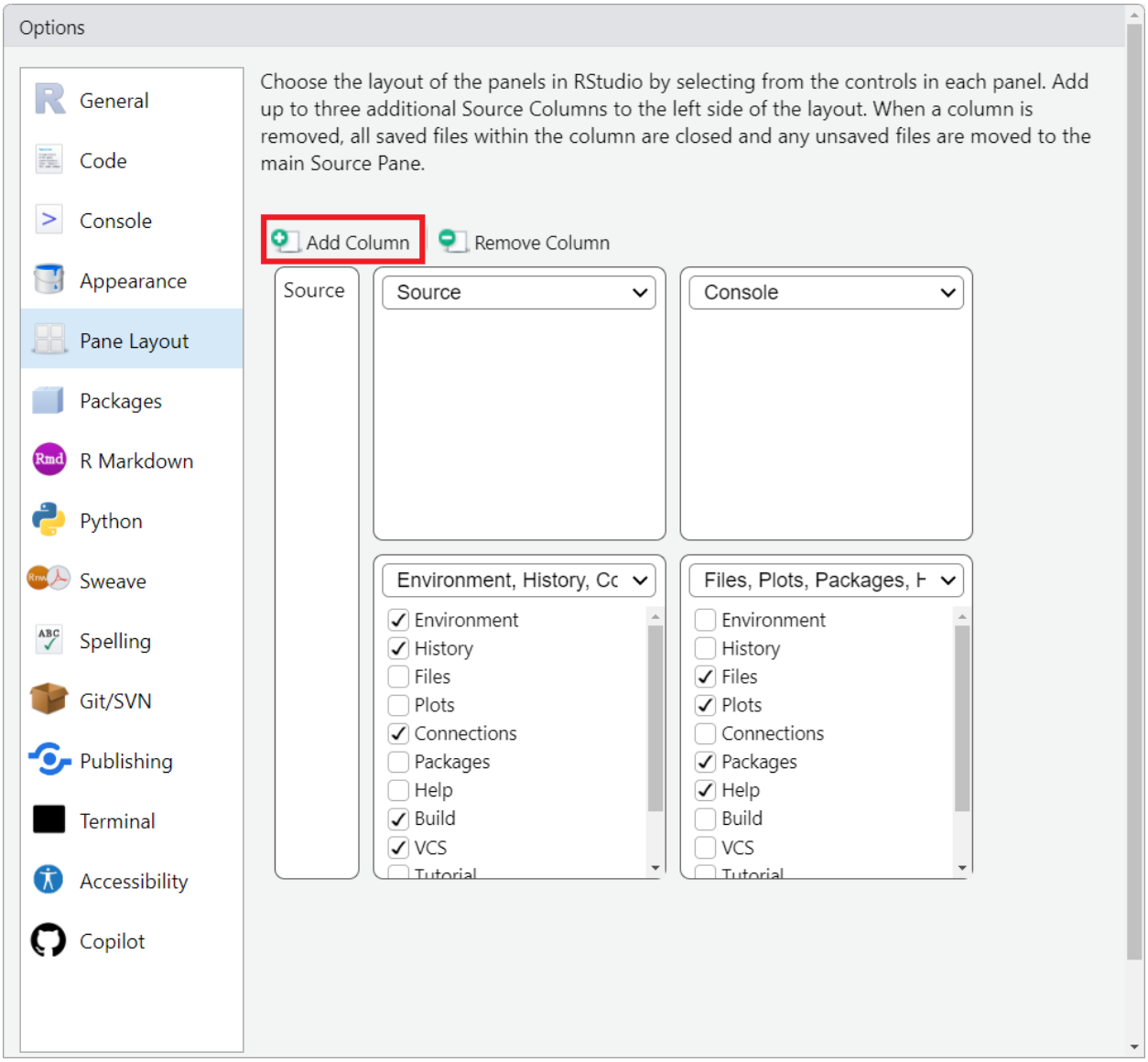
(2) The R code you are translating to. You will create a new R code file here, paste the results from ChatGPT, and fix it up here.

(3) The R console. Code you run in R that produces printed output will appear here.

CREATING A THREE-PANEL LAYOUT

So how to do get the three-panel layout above? Take the following steps:

1. In RStudio, go to Tools -> Global Options.
2. On the Options screen, select “Pane Layout”.
3. On the “Pane Layout” panel, push the “Add Column” button. Arrange the other panels to your liking.



4. Press “OK” at the bottom of the Options Screen.
5. When you get back to the RStudio main screen, you should have the three-panel layout shown above.

SAS CODE TO TRANSLATE

Now let’s get translating! We will start with a very simple SAS program. This SAS program joins two datasets and generates a table of frequencies. Here is the SAS code:

```
* Create Colors Dataset;
data Colors;
input Category Color $ Count;
datalines;
1 red    23
1 red    11
1 orange 50
1 orange 23
```

```
2 green  18
2 green   5
2 blue    3
2 blue   44
;

* Create Categories Dataset;
data Categories;
input Category Label $;
datalines;
1 Warm
2 Cool
;

* Join datasets;
data ColorCategory ;
merge Colors Categories;
by Category;
run;

* Get frequencies;
proc freq data=ColorCategory ;
tables Category*Color;
weight Count;
title 'Color Counts';
run;
```

The output from the above program is as follows:

Color Counts

The FREQ Procedure

Frequency
Percent
Row Pct
Col Pct

Table of Category by Color					
	Color				
Category	blue	green	orange	red	Total
1	0 0.00 0.00 0.00	0 0.00 0.00 0.00	73 41.24 68.22 100.00	34 19.21 31.78 100.00	107 60.45
2	47 26.55 67.14 100.00	23 12.99 32.86 100.00	0 0.00 0.00 0.00	0 0.00 0.00 0.00	70 39.55
Total	47 26.55	23 12.99	73 41.24	34 19.21	177 100.00

INITIAL PROMPT

Let's start with a minimal prompt. We will use "Translate the following SAS code into R:", and then paste the SAS code into the ChatGPT input box. Here is what the AI tool did:

r Copy

```r
# Create the Colors data frame (equivalent to the SAS "Colors" dataset)
Colors <- data.frame(
  Category = c(1, 1, 1, 1, 2, 2, 2, 2),
  Color    = c("red", "red", "orange", "orange", "green", "green", "blue", "blue"),
  Count    = c(23, 11, 50, 23, 18, 5, 3, 44)
)

# Create the Categories data frame (equivalent to the SAS "Categories" dataset)
Categories <- data.frame(
  Category = c(1, 2),
  Label    = c("Warm", "Cool")
)

# Merge the two data frames by Category (like the SAS MERGE statement)
ColorCategory <- merge(Colors, Categories, by = "Category")

# Create a weighted frequency table for Category by Color using xtabs.
# The Count variable is used as weights.
freq_table <- xtabs(Count ~ Category + Color, data = ColorCategory)

# Print the title and the frequency table, similar to the SAS proc freq output.
cat("Color Counts\n")
print(freq_table)
```

As you can see, on the first pass, the model used Base R functions. The code is commented nicely, and mentions the equivalent SAS code. The datasets are converted to data.frames. The `merge()` function was used to join the two datasets, and the `xtabs()` function was used to get the frequencies. Not bad!

If you copy and paste this code into RStudio and run it, you will see the results in the console:

```
Category blue green orange red
       1    0     0     73  34
       2   47    23      0   0
```

The counts match!

However, there are some improvements we can make to this code. These improvements begin the process of translation revision.

FIRST REVISION

An important thing to realize is that our chat is actually a running conversation. ChatGPT will remember all your previous prompts, and you can talk to the chatbot just like you would talk to a junior programmer. That means if you want to make a revision, you can just type the desired revision into the prompt dialog.

For our translation project, most R programmers prefer to use Tidyverse instead of Base R. So we can revise our code by simply making the request: "Use Tidyverse instead of Base R". Here is what it did:

r Copy

```r
# Load the required tidyverse packages
library(tidyverse)

# Create the Colors data frame as a tibble
Colors <- tibble(
  Category = c(1, 1, 1, 1, 2, 2, 2, 2),
  Color    = c("red", "red", "orange", "orange", "green", "green", "blue", "blue"),
  Count    = c(23, 11, 50, 23, 18, 5, 3, 44)
)

# Create the Categories data frame
Categories <- tibble(
  Category = c(1, 2),
  Label    = c("Warm", "Cool")
)

# Merge the two data frames by Category using left_join
ColorCategory <- Colors %>%
  left_join(Categories, by = "Category")

# Create a weighted frequency table by grouping and summing the Count variable
freq_table <- ColorCategory %>%
  group_by(Category, Color) %>%
  summarize(WeightedCount = sum(Count), .groups = "drop")

# Print the title and the frequency table
cat("Color Counts\n")
print(freq_table)
```

It worked! The code has been refactored to use Tidyverse instead of Base R. Here are the results:

```
# A tibble: 4 × 3
  Category Color  WeightedCount
     <db1> <chr>          <db1>
1        1 orange            73
2        1 red               34
3        2 blue              47
4        2 green             23
```

SECOND REVISION

Looking at the above results, we can see that the “Category” is still using the number, instead of the label. Therefore, we will simply ask ChatGPT to revise it: “In the frequency table, use the Category Label instead of the Category number.”

Also notice that we are missing the percents for each category. So, we will add to the prompt: “Also add the percentages for each Category/Color combination. Use the total WeightedCount as the denominator.”

Here is the response:

```r
library(tidyverse)

# Create the Colors data frame as a tibble
Colors <- tibble(
  Category = c(1, 1, 1, 1, 2, 2, 2, 2),
  Color    = c("red", "red", "orange", "orange", "green", "green", "blue", "blue"),
  Count    = c(23, 11, 50, 23, 18, 5, 3, 44)
)

# Create the Categories data frame as a tibble
Categories <- tibble(
  Category = c(1, 2),
  Label    = c("Warm", "Cool")
)

# Merge the two data frames by Category so that the label is available
ColorCategory <- Colors %>%
  left_join(Categories, by = "Category")

# Create a weighted frequency table with percentages using the overall total weighted count as the
freq_table <- ColorCategory %>%
  group_by(Label, Color) %>%
  summarize(WeightedCount = sum(Count), .groups = "drop") %>%
  ungroup() %>%
  mutate(Total = sum(WeightedCount)) %>%
  mutate(Percentage = 100 * WeightedCount / Total) %>%
  select(Label, Color, WeightedCount, Percentage)

# Print the title and the frequency table
cat("Color Counts\n")
print(freq_table)
```

Nice! The prompt worked perfectly! Here is the output:

```
# A tibble: 4 × 4
  Label Color  WeightedCount Percentage
  <chr> <chr>          <dbl>      <dbl>
1 Cool  blue              47       26.6
2 Cool  green             23       13.0
3 Warm  orange            73       41.2
4 Warm  red               34       19.2
```

SUBSEQUENT REVISIONS

Keep iterating until you are happy with the translation. When you are happy with the translation, press the "Like" button. This feedback will let ChatGPT know that this is how you want future translations to be done. From then on, ChatGPT will remember this feedback and try to translate things according to the instructions. That means subsequent translations will go much faster, with fewer revisions.

TIPS AND TRICKS FOR BETTER TRANSLATION

As can be seen in the examples above, ChatGPT is actually quite intelligent when it comes to understanding and translating SAS and R code. The above examples required no hand editing at all. Of course, these were very simple examples. For real life translations, your programs will be more complicated, and likely require some hand editing of the code.

Having performed many translations of this type, the authors have discovered some techniques that can improve the quality of the translation, and therefore reduce the number of manual edits. Here is a summary of our findings:

Be very specific in the prompt

If you want the tool to use specific R packages in the translation, state those packages directly in the prompt. If you want the tool to use specific functions, state those functions directly in the prompt.

Use popular packages

The research conducted observed that AI tools do best with popular packages. The reason is simply that there is a lot more code on the internet from which the AI tools can learn. For example, if you are doing a lot of data manipulation, translation using **dplyr** comes out very well.

Provide Positive Feedback

If the tool provides a good translation, give it a "thumbs up" for positive feedback. The tool will then incorporate that feedback into future responses and thereby improve subsequent translations. This feedback allows the translation to get better and better over time within the same chat.

Train on Key Packages

It is helpful to train the tool on key packages first and then work your way up to complete programs. For example, if you wanted to use **r2rtf** for reporting, it is best to create some narrow examples simply translating PROC REPORT to **r2rtf**. Focusing on a small program allows you to adjust the translation, and make sure the tool understands how you want the **r2rtf** calls to come out. Once that is accomplished, you can then translate a complete program, and your PROC REPORT code will be translated as desired.

Teach it what you want

If a translation is not exactly perfect, tell the tool how to fix it. You can be very specific: "Add x parameter to y function", "Use X function instead of Y function", etc. Remember to provide good feedback if it performs as instructed. The tool will then remember these instructions, and use them in future translations without prompting. This technique is especially valuable if you are using a package which is not so popular.

Use references

ChatGPT and other AI tools have access to the internet, and will refer to specific papers, web sites, chats, blog posts, or other documentation if you tell it. You can give a prompt like "Use X function as described on Y blog post [provide link]". The tool will then go read the blog post, try to understand it, and use the syntax as described on that post.

Provide Corrections

Finally, if the tool does not provide a good translation for a section or function, you can simply supply the desired code: “Replace X section with this: [paste code]”. Again, if the tool does what you want, give it positive feedback. These corrections and feedback will contribute to better translations in the future.

LIMITATIONS OF MACHINE TRANSLATION

Prior research also discovered the following limitations on the use of AI tools:

Hallucinations

A “hallucination” is when the AI tool basically makes things up. With past “predictive” models, these hallucinations were rather frequent. With the advent of reasoning models, the number of hallucinations has gone down considerably. They can, however, still occur. Occasionally the chatbot will make up functions or provide non-sensical syntax. One must therefore be vigilant, and always test your code and compare the output.

Absence of feedback causes degradation

The AI tool is trying to “please” you. If it gives you a good response, but you don’t provide positive feedback, it may assume that response is not good, and try to come up with something else. We have observed that if you continue to not give the tool feedback, the responses will become progressively worse. Therefore, one must be attentive to providing good feedback to ensure quality responses.

Inconsistent responses

We have noticed that AI tools will rarely give the exact same response to the exact same question. That is to say, the same question can generate inconsistent responses. This inconsistency is also true of code translation. The tool will not usually translate the same piece of code in the same way. Developers must take this into account when performing machine translation.

Memory is by session

As described above, corrections and feedback can improve the quality of the translation. This improvement is made possible by a “memory” feature in ChatGPT. It must be noted that the memory feature is local to the user and the chat. That means the “memory” and any learnings will not be available to other users, or other chats. Therefore, it is good to reuse chats if you have done a lot of work teaching ChatGPT what you want.

Gives up on long code

The authors have observed that AI translation tools will sometimes give up on long sections of code, or leave out big sections. That is to say, the AI tools do well when the code is relatively short: 200 lines or less. For long programs, it is best to chunk up the program and translate piece by piece. Then paste everything together into a single program.

Gives up on hard code

The authors also have observed that AI translation tools will sometimes give up on hard sections of code, or tricky logic. The tools do extremely well with straight-forward code. Very dynamic code or obstruse logic can confuse the tool. For complicated sections of code, the tool will sometimes just skip over it and not translate it at all. Or leave a comment for the user, but not translate the section.

In some ways, giving up is an improvement over earlier versions of ChatGPT. In earlier versions, the tool would simply start making stuff up. The new anti-hallucination routines make the LLMs a little more conservative, which is good. It does mean, however, that sometimes there are holes in the translation. You can help the tool by giving it suggestions on how to do the translation for those sections, providing reference material, or providing actual code snippets it can learn from.

CONCLUSION

Based on prior research, the authors have determined that code conversion and generation involving SAS & R using ChatGPT is feasible and can result in significant time/cost saving. The latest versions of ChatGPT can in fact produce very high-quality translations. You can talk to the chatbot like a junior programmer, providing feedback, giving suggestions, and generally leading it in the direction you want. The reasoning style models are far superior to previous models. They do a much better job with coding tasks, and are vastly preferred over prior models. With some patient understanding of how to use this technology, it is indeed possible to have ChatGPT help you translate from SAS to R in a much more efficient way.

REFERENCES

Mollick, Ethan. 2024. Co-Intelligence: Living and Working with AI. New York: Portfolio/Penguin.

CONTACT INFORMATION

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

David J. Bosak
Archytas Clinical Solution
dbosak01@gmail.com

Brian Varney
Experis
brian.varney@experis.com