PhUSE US Connect 2024
Paper ET08

AI Exploration and Innovation for the Clinical Data Scientist

Phil Bowsher, Posit; Boston, USA
Cole Arendt, Posit, Boston, USA
Bo Ci, Genentech, San Francisco, California
Vincent Shen, Roche, Basel, Switzerland

Abstract

This paper will cover some use cases of using AI in the clinical trials space. The tools and use cases below are exploratory in nature and offer insight and information on using AI in the clinical domain focused primarily on Generative AI and how to be a more productive coder. Generative AI is a category of AI models and tools designed to create new content. The focus of this paper will be on generating information/code and using AI as a productivity aid in programming.

Using GenAI to aid in programming is a mix of providing context of the objective to help achieve better output. This can be aided by inserting a well-written description of the programming objective and iterating with more specific tasks. For an overview, be sure to watch “A hacker's guide to open source LLMs” from the posit::conf(2023) https://www.youtube.com/watch?v=sYliwvml9Es

Introduction

This paper is for clinical statistical programmers that are wondering how AI can be applied for clinical work. Rather than focusing on applications of AI in drug discovery, this paper will highlight tools that can be used directly by Clinical Data Scientists and programmers. We will break these tools into the sections below:

I. AI Pair Programmer Tools
II. Large Language Models
    A. Public & Local
    B. Genentech Example

A. GenAI to Enhance Your Statistical Programming

AI can be used to enhance open-source statistical programming. Below we will discuss GenAI to support programmers in the process of writing code. There are many GenAI IDE tools such as JupyterAI. VS Code has many extensions for Generative AI, including Codeium, AWS Code Whisperer, Tabnine etc.

1. GitHub Copilot

GitHub Copilot is a cloud-based artificial intelligence tool developed by GitHub and OpenAI to assist users in various development environments like RStudio and Visual Studio (VS) Code. It works as a generative AI tool used to generate code. Copilot helps users by autocompleting code using AI to suggest entire functions in real-time within the IDE. The coding suggestions are based on the programmers project's context and related files. Copilot suggestions are via “ghost text” based on the context of the code and provided via API endpoint.

GitHub Copilot requires a monthly fee or yearly subscription. It also requires that the user have an active GitHub Copilot subscription. Subscriptions will need to be managed via GitHub.com Individual plan or managed through an employer's Microsoft plan.

Below we will discuss how to get up and running with GitHub Copilot in both VS Code and RStudio…

Copilot in VS Code

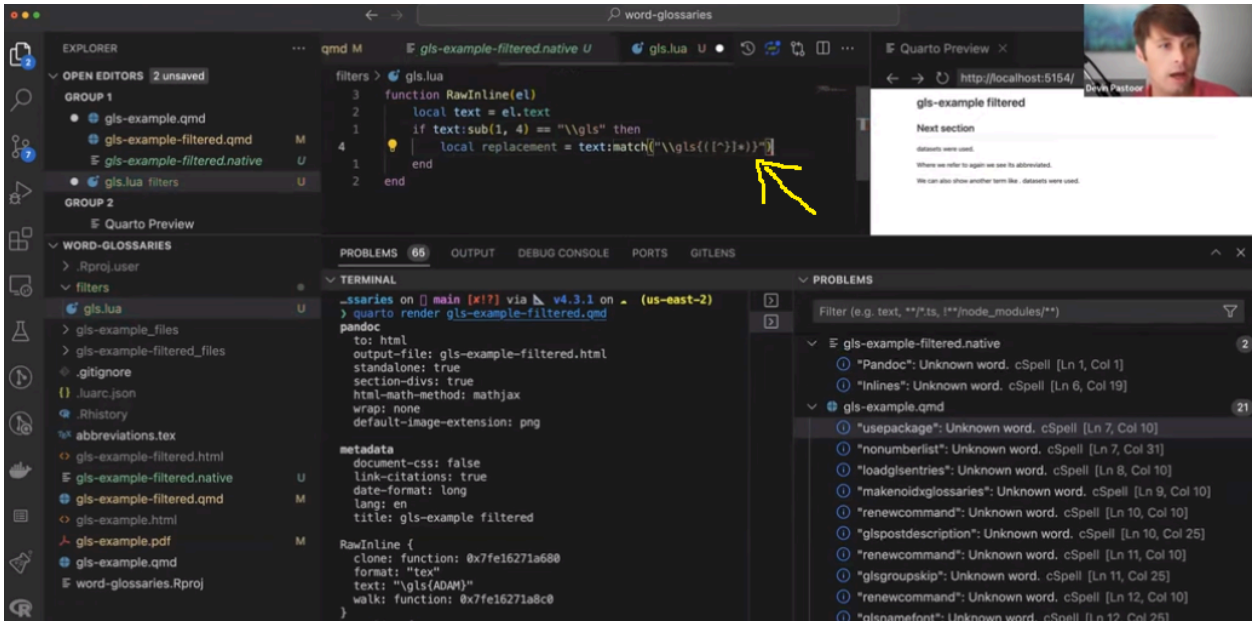
To use GitHub Copilot, you must first install the GitHub Copilot extension. You can download VS Code for a desktop machine here:

https://code.visualstudio.com/

Below is an example of using VS Code with Copilot.

- Install GitHub Copilot Extension - https://marketplace.visualstudio.com/items?itemName=GitHub.copilot

You can watch Devin Pastoor programming with Copilot live in VS Code during the R in Pharma workshop **Quarto All the Things!**
https://youtu.be/k-dQ36sx4Rk?t=8276



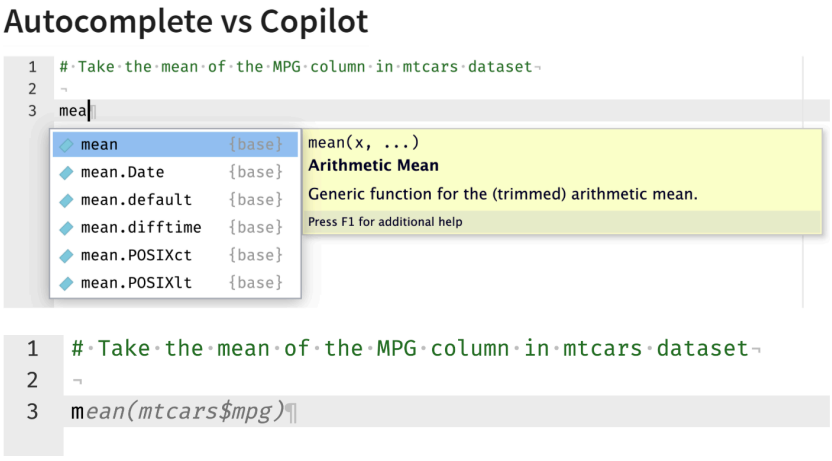
You can read more about Copilot in VS Code here:
https://code.visualstudio.com/blogs/2023/03/30/vscode-copilot
https://code.visualstudio.com/docs/editor/github-copilot
https://docs.github.com/en/copilot/using-github-copilot/getting-started-with-github-copilot

Copilot in RStudio

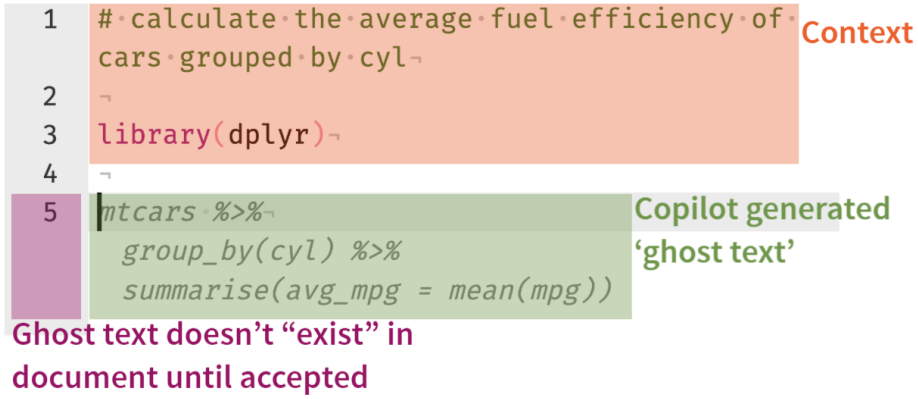
A feature request came in through Github to add Copilot to RStudio. Issue #10148 was officially added in September 2023 after receiving over 500 upvotes.
https://github.com/rstudio/rstudio/issues/10148

Tom Mock gave a talk introducing Copilot into RStudio. You can view his slides here:
https://colorado.posit.co/rsc/rstudio-copilot/

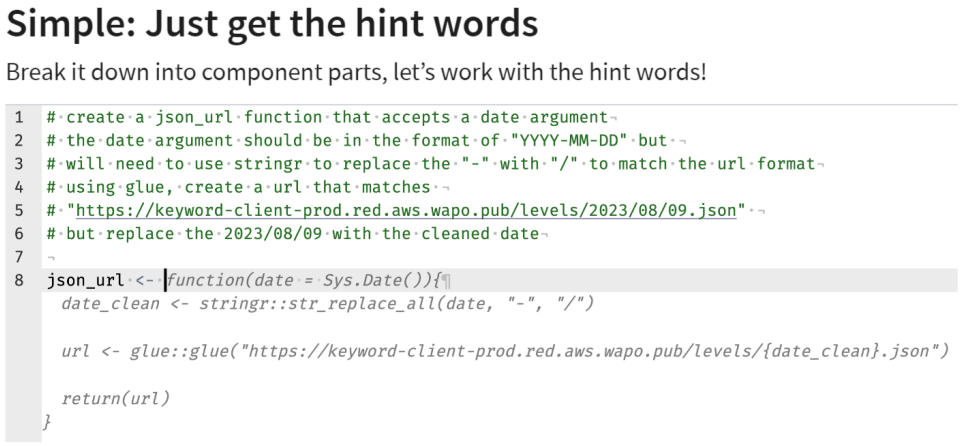
In his talk, he shows the difference between autocomplete and Copilot as:



Below is Copilot generated Code as seen in the video with Devin above.



Below you can see the context added to the top of the script:



You can download RStudio for a desktop machine here:
https://posit.co/download/rstudio-desktop/

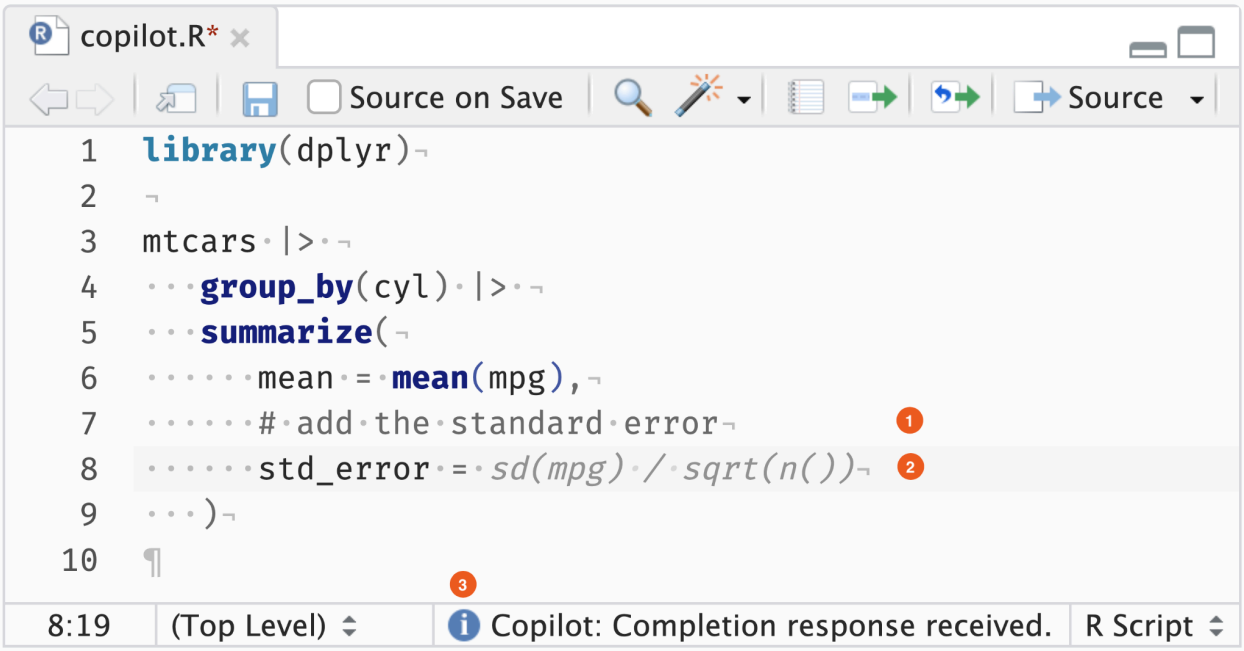
To get started with Copilot in RStudio, follow the steps in the document below:

https://docs.posit.co/ide/user/ide/guide/tools/copilot.html

To enable GitHub Copilot in RStudio:

- Navigate to Tools > Global Options > Copilot.
- Check the box to “Enable GitHub Copilot”.
- Download and install the Copilot Agent components.
- Click the “Sign In” button.
- In the “GitHub Copilot: Sign in” dialog, copy the Verification Code.
- GitHub Copilot: Sign in
- Navigate to or click on the link to https://github.com/login/device, paste the Verification Code and click “Continue”.
- GitHub will request the necessary permissions for GitHub Copilot. To approve these permissions, click “Authorize GitHub Copilot Plugin”.
- After the permissions have been approved, your RStudio IDE will indicate the currently signed in user.
- Close the Global Options dialogue, open a source file (.R, .py, .qmd, etc) and begin coding with Copilot!

Below is Copilot working in RStudio:



The above orange numbers represent:

1. A simple but specific comment providing additional context to Copilot.
2. Copilot’s code suggestion, shown in light gray “ghost text”.
3. The Copilot status bar, which indicates whether RStudio is waiting on a response to be generated, a completion response has been received, or no completions are available.

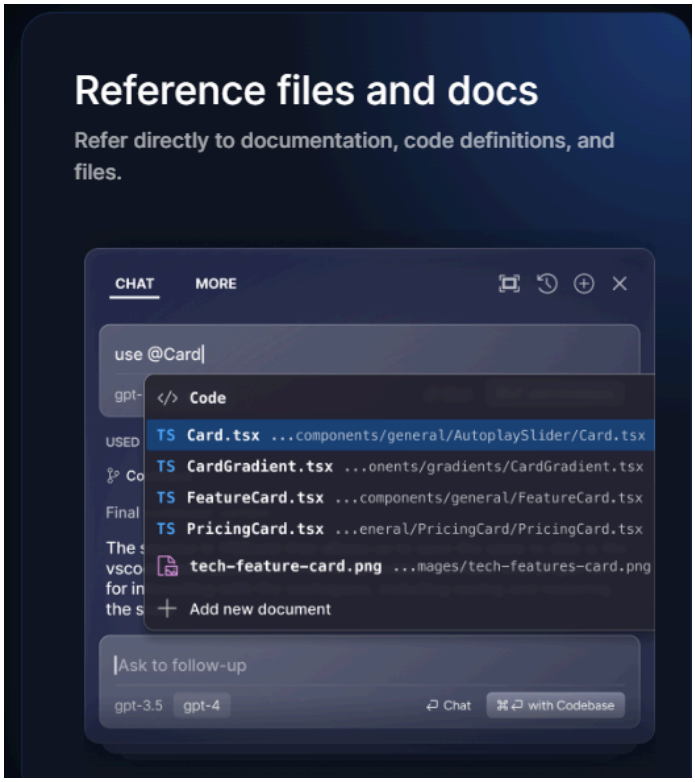
Users can use Posit Cloud as an online platform for learning R. Posit Cloud also supports GitHub Copilot, and users can experience AI code assistance within your RStudio projects on Posit Cloud. More information about this can be found here:
https://posit.co/blog/github-copilot-on-posit-cloud/

Example with cursor.sh
LLMs are popular for generating code from natural language for tasks like autocompleting and summarization. While helpful for some use cases, there are 2 popular topics that arise: Stochastic Parrots & Law of Averages

Stochastic Parrots was coined by Emily M. Bender in the 2021 artificial intelligence research paper "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?". Stochastic Parrots repeat back the contents of the trained data and therefore don't understand the problem addressed or know if it is incorrect etc. This can be a challenge for a beginner programmer for example. You can read more here: https://en.wikipedia.org/wiki/Stochastic_parrot

Law of averages states that outcomes revert to the mean (https://en.wikipedia.org/wiki/Law_of_averages). LLMs often provide responses based on its training data with responses that can lack industry context or are just made up. LLMs are trained on data at a point in time. For example, Shiny for Python came out in July of 2022, after the training point for ChatGPT. If you request code for an example app, ChatGPT will likely as of this writing tell you Shiny is a web framework for R and suggest Dash or another tool. These "average" responses may not be of high enough quality to provide value to specific questions. You can read more about these LLM challenges here: https://hbr.org/2023/02/generative-ai-wont-revolutionize-search-yet

A popular AI-first Code Editor is cursor.sh that helps combat these challenges. cursor.sh was designed for pair-programming with AI, where it can read current documentation, review code, and debug. Below shows an example of referring to specific documentation:



https://cursor.sh/

Cursor.sh provides RAG support for supplementing LLMs with further context while enhancing the programming experience in an IDE (similar to Visual Studio Code). For example, a programmer can feed cursor.sh the Shiny for Python guide, which then will provide working code when requested.

2. ChatGPT

ChatGPT (or Chat Generative Pre-Training Transformer) was released to the public in March 2023 and provides insights through interplay in a conversational way. There are many ways to use ChatGpt without having to exit your Integrated Development Environment (IDE) of choice.

ChatGPT in VS Code

There are many ChatGPT extensions for VS Code that help provide ChatGPT's features. Below are a few of the popular VS Code extensions:

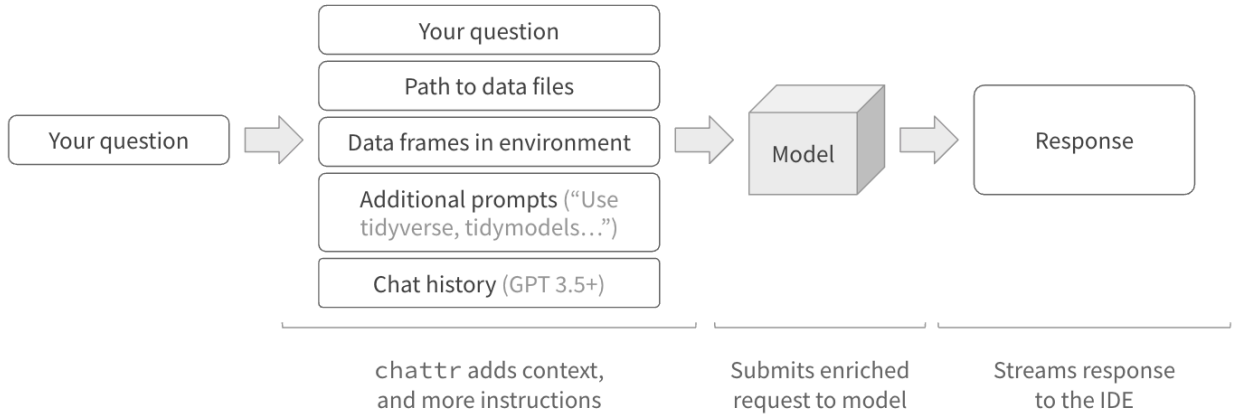
1. Visual chatGPT Studio - https://marketplace.visualstudio.com/items?itemName=jefferson-pires.VisualChatGPTStudio
2. ChatGPT - https://marketplace.visualstudio.com/items?itemName=timkmecl.chatgpt
3. Several more (Codium, CodeGPT, Genie AI, etc.) as discussed here:
    a. https://www.guidingtech.com/best-chatgpt-extensions-for-vs-code/

ChatGPT in RStudio

Like VS Code, there are many ways to use ChatGPT in RStudio. Below we will feature the package {chattr}:

A. chattr

chattr is an interface to LLMs (Large Language Models). It enables interaction with the model directly from the RStudio IDE. There are 2 ways to use chattr, either by submitting a prompt to the LLM in a script, or by using the provided Shiny Gadget with RStudio. You can read mre and see how it works here:
https://github.com/mlverse/chattr



Below we will review these 2 methods:

1. ChatGPT with chattr

ChatGPT offers A REST API endpoint that can be used to communicate with their LLM. OpenAI requires a secret key to authenticate. To obtain your secret key, follow this link:
https://platform.openai.com/account/api-keys

chattr will look for the secret key inside the Environment Variable called OPENAI_API_KEY. To set this variable, the usethis package can be used:

```
usethis::edit_r_environ()
```

Then add your key to the file and call it OPENAI_API_KEY. Be sure then to save the file and restart R.

Use the chattr_test() function to confirm that your connection works.

```
> chattr::chattr_test()

── Testing chattr
● Provider: Open AI - Chat Completions
● Path/URL: https://api.openai.com/v1/chat/completions
● Model: gpt-4
✔ Connection with OpenAI cofirmed
|--Prompt: Hi!
|--Response: Hi! I see that you have specific requirements for the code. Could you
please provide more details on what you would like the code to do?
```

Local Models with chattr

Some people may not have access to paid or public APIs or organizations do not allow employees to use public LLMs such as OpenAI's GPT. One option is to use a local model on your desktop, laptop or server. Below we will review local models. This information is for personal use and be sure to review the license information. This will prevent data being sent outside of your organization. One challenge is that the smaller the LLM model, the less effective it may be so performance testing is advised.

For our example below, we are going to use LlamaGPTJ-chat, a C++ tool that runs locally to integrate with local LLMs. Another option for local models is Ollama (https://ollama.ai/library).

The first step is to install LlamaGPT-Chat through the "compiled binary" that is specific to your Operating System: https://github.com/kuvaus/LlamaGPTJ-chat/releases

Second, download a model file here:
https://github.com/kuvaus/LlamaGPTJ-chat?tab=readme-ov-file#gpt-j-llama-and-mpt-models

Currently GPT-J, LLaMA and MPT models are supported. There are various organizations that provide trained models. Below we will outline how to use a GPT-J model that is provided by Nomic AI via:
https://gpt4all.io/

You can download the model at:
https://gpt4all.io/models/ggml-gpt4all-j-v1.3-groovy.bin

Save the model to your computer/server. Then run the following in R:

```
Unset
library(chattr)

chattr_use("llamagpt")
```

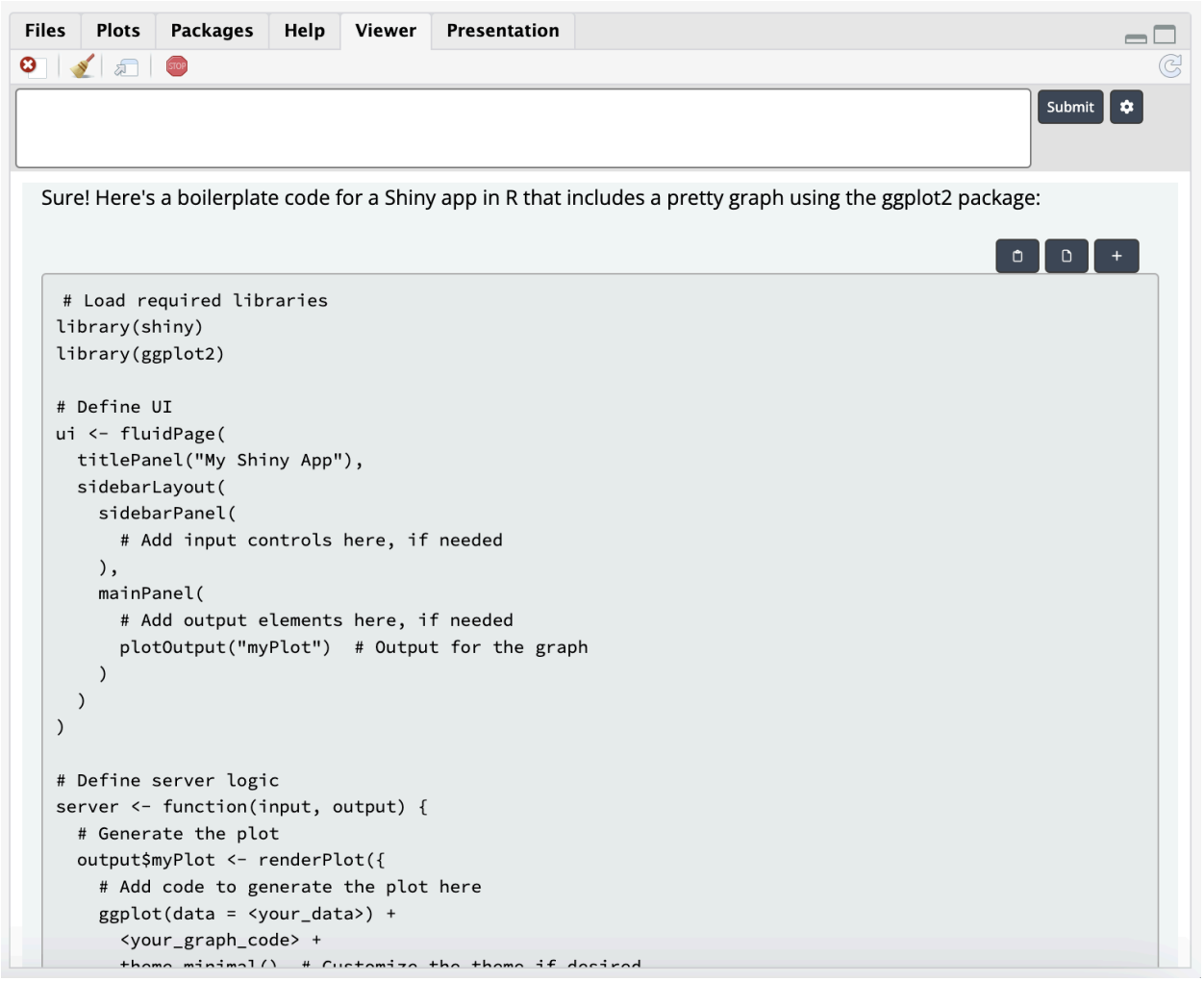
META AI also released LLaMA and it can be used as a backend model. On common benchmarks, LLaMA outperforms OpenAI's GPT-3, as a smaller model.

https://blogs.rstudio.com/ai/posts/2023-05-25-llama-tensorflow-keras/

You can apply for access to LLaMA here:

https://ai.facebook.com/blog/large-language-model-llama-meta-ai/

The LLaMA models are around 4.2 Gb to 8.2 Gb each. You can also use derived models such as Vicuna or ggml snoozy. If needed, you can specify the Path/URL and model by updating the path, and model arguments in chattr_defaults().



Exploring Ideas and Strategies for Using Local LLMs on Your Internal Data

Many publicly available LLMs, either through a service or local, will likely lack the industry context for specific drug discovery/development use cases. Also, many organizations limit the use of public LLMs. Some organizations allow public AI services but prevent local data from being sent. One question often is if local LLMs can be used to create a private chat tool that interacts with behind the firewall local documents. By augmenting pre-existing models with your

use case specific data, organizations hope to create a tool focused on specific tasks. **Some offerings provide services to private cloud environments like Azure OpenAI subscriptions** with tools such as FractalGPT:
https://azuremarketplace.microsoft.com/en-us/marketplace/apps/neal_analytics.fractal_gpt?tab=overview

The monthly subscription can run $5-$10,000 per month.

When discussing augmenting LLMs, there are 2 areas often discussed: **RAG** (Retrieval Augmented Generation) and **Fine-tuning**. Sometimes the methods are combined and not mutually exclusive.

RAG is usually used in leveraging new sources of information to enhance a LLM. It was introduced by Meta in 2020 here:
https://arxiv.org/abs/2005.11401v4

With RAG, documents can be stored as embeddings in vector databases and made available to a prompt. With RAG, there is no extra training required as the responses are augmented with information from a retriever like Azure AI Search. See below sample examples with Azure:
https://github.com/Azure-Samples/azure-search-openai-demo

Another example of RAG is the Databricks Data Intelligence Platform. Organizations can host RAG and Generative AI applications that use proprietary data stored in a Lakehouse. You can watch an example here:
https://www.youtube.com/watch?v=89CTQQzpe1U

There are also open-source tools like:
https://github.com/llmware-ai/llmware

Fine-tuning often involves taking an existing (pre-trained) model and feeding it new (often domain) data to update the weights with the new input. Some API services enable fine-tuning with your imputed data like OpenAI API:

https://platform.openai.com/docs/guides/fine-tuning

There are costs associated with fine-tuning such as in the time and effort to prep the data and computational resources. Fine-tuning models requires experienced machine learning progressions and engineers.

An example is Clinical Camel from the University of Toronto, an LLM explicitly tailored for clinical research. It was Fine-tuned from LLaMA-2 using QLoRA.
https://arxiv.org/abs/2305.12031

There are various places to find existing models like the Hugging Face community. Hugging Face has guidance on fine-tuning models here:
https://huggingface.co/docs/transformers/training

Some organizations for example will fine-tune Hugging Face models like Zephyr using Azure AI Studio and infrastructure with infrastructure tools like Kubernetes etc.:
https://huggingface.co/HuggingFaceH4/zephyr-7b-beta

Vendors have also create models for various commercial and drug discovery activities like:
https://catalog.ngc.nvidia.com/orgs/nvidia/teams/clara/models/megamolbart
https://www.mosaicml.com/mpt

There are also open-source tools and interfaces for fine-tuning LLM like:

https://github.com/stochasticai/xturing
https://github.com/OpenAccess-AI-Collective/axolotl

For context, an example with steps are below:

Biomedical Question Answering over Custom Datasets with LangChain and Open Source LLMs on Hugging Face:
https://github.com/databricks-industry-solutions/hls-llm-doc-qa
https://notebooks.databricks.com/notebooks/HLS/hls-llm-doc-qa/index.html#hls-llm-doc-qa_1.html

Below is some **experimental information on ideas** for using locally stored documents that reside on a computer. You can read more here:
https://bdtechtalks.com/2023/06/01/create-privategpt-local-llm/

As described above, there are many local LLMs (Dolly, Vicuna, GPT4All, llama.cpp & various Hugging Face models) that can run on a local computer. There are also Clinical Language Models such as Clinical Camel and gatortron. Gatortron is from the University of Florida and NVIDIA that has been trained with healthcare data such as real-world EHR data. You can read more about it here:
https://huggingface.co/UFNLP/gatortron-base

There are other components required to set up a LLM to ingest local documents such as a Vector database (DuckDB, Faiss, Pinecone, Hopsworks, MongoVcore etc.). One project that bundles these components together is the Python based PrivateGPT:
https://github.com/imartinez/privateGPT

PrivateGPT supports a variety of file types such as PDF and Word etc. Below are the steps that build on the information from above:

Install PrivateGPT on the local machine with:

```
git clone https://github.com/imartinez/privateGPT.git
cd privateGPT/
pip install -r requirements.txt
```

Using either GPT4ALL-J or llama.cpp from above, create a “models” folder in the PrivateGPT directory and add the model file. Copy local files to the “source_documents” folder. Next run python ingest.py to populate your vector database with the embedding values (LangChain and SentenceTransformers from Hugging Face, OpenAI, Cohere, AI21 Labs, as well as open source models etc.) from the added files. An internet connection is required for LangChain. After it is completed, execute “python privateGPT.py” to run the model.

Lastly, create the User interface (UI). For this section, we will highlight Shiny which is great for rapidly prototyping applications. With the UI you have various options with Shiny such as using Shiny via R, Shiny with Reticulate, or Shiny for Python. Below is an article detailing how to make streaming AI chat apps with Shiny for Python:
https://shiny.posit.co/blog/posts/shiny-python-chatstream/

There is a talk that explains the Shiny work here:

AI and Shiny for Python: Unlocking New Possibilities - posit::conf(23)
https://www.youtube.com/watch?v=lXOMJvuPN_Y

The app above uses OpenAI API. Here is some sample code using OpenAI:
https://github.com/wch/chatstream/blob/main/examples/doc_query/app.py

It is worth noting that OpenAI is a remotely provided service, so every request will send data outside of the organization. This is not a great fit in every scenario. In other cases, when data or licensing requirements restrict data movement and privacy, it is important to find another LLM deployment strategy.

One is to store the GPT API locally as discussed above. The other way is to centralize the model deployment using a REST API or other generic interface. This is one of the things that OpenAI has pioneered well - deploying different models with consistent APIs so that it is easy to move between different generations. If you wrap the model in an API framework like Plumber, Flask, or FastAPI, then generic API deployment tools and frameworks apply. For example, we have used Posit Connect with Off-Host-Execution to privately expose an LLM over a REST API interface while running on GPUs in AWS.

This type of deployment paradigm allows you to create your own OpenAI type of interface - where models are hosted, centralized, and interchangeable within your organization.

If you are interested in other areas of machine learning, a simpler app is here:
https://github.com/philbowsher/Workshop-R-Tensorflow-Scientific-Computing/tree/master/part2/keras_image_recognition

It uses a pre-built model for image recognition.

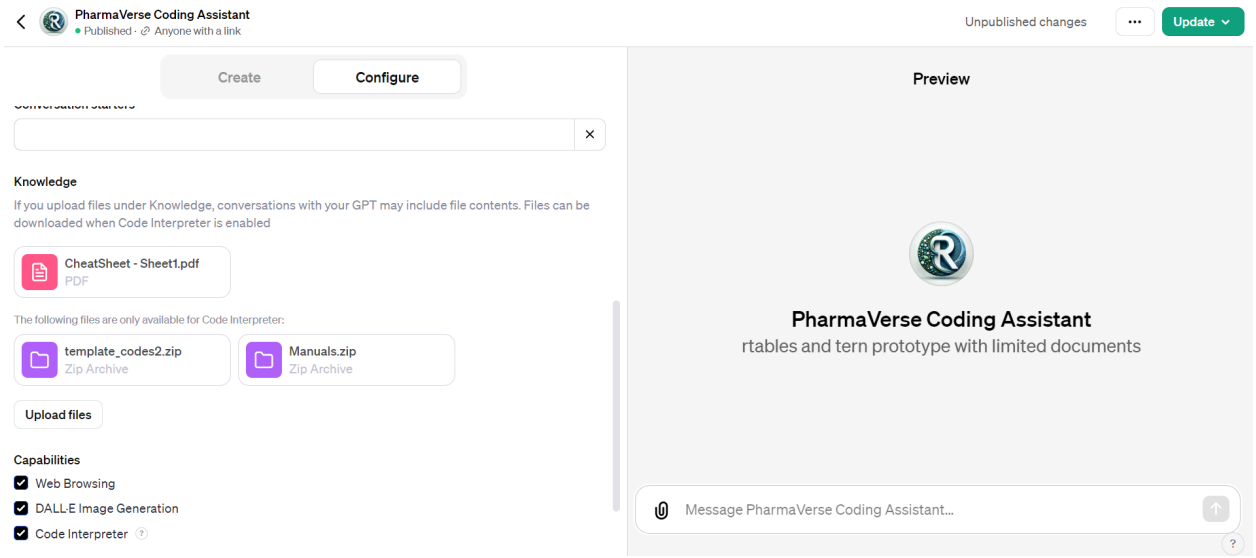
Industry Example - Roche/Genentech

Bo Ci (Roche/Genentech) recently discussed the proof-of-concept (PoC) projects at Roche for using a large language model (LLM)-based framework to develop R codes for producing the outputs from ADaM datasets.

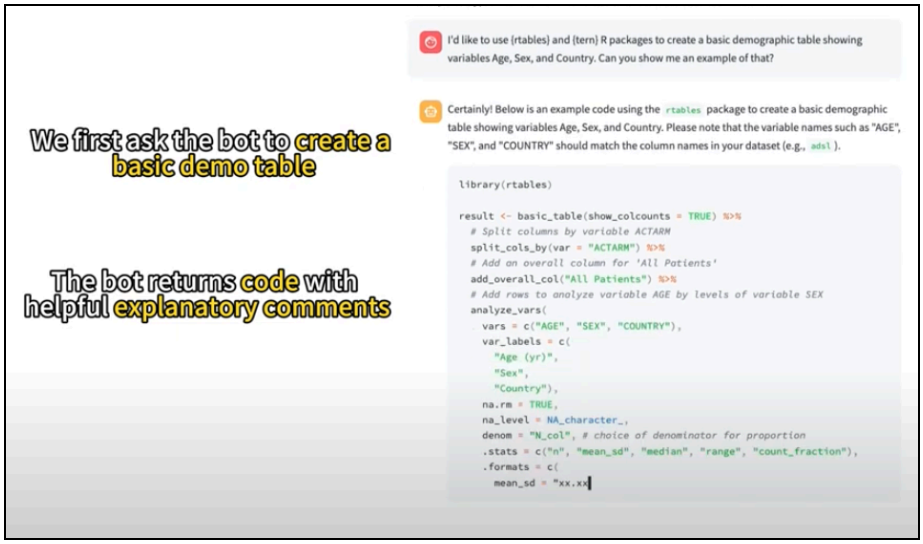
One PoC of leveraging the GPT-4 Code Interpreter with supporting files (template codes, variable dictionary and function manuals) demonstrated a good potential of completing following tasks per user's natural language requests:
1) select the fit-for-purpose template code
2) search in the variable dictionary and propose variables to use
3) modify template codes to filter patients and update the output contents

Bo's presentation (entitled "Exploration of Large language model-based frameworks to develop R codes for output delivery in pharma") can be found here:
https://www.youtube.com/watch?v=HyDvK5Fikfs

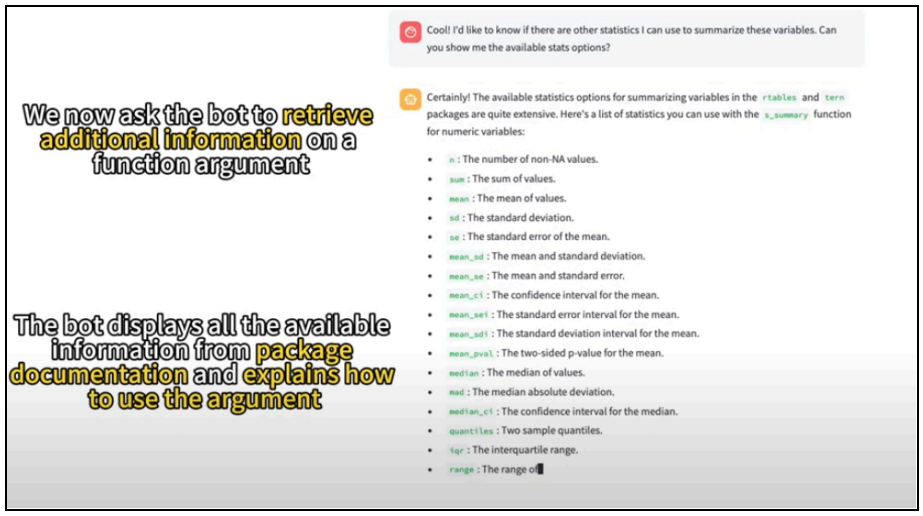
As OpenAI released the new feature GPTs last year, PharmaVerse Coding Assistant GPT was created as another PoC via providing documents to the Knowledge (Left middle in the screen copy below). The advantage of this approach is that users don't need to upload documents every time themselves and the team could implement a better centralized control of the documents via updating the Knowledge.

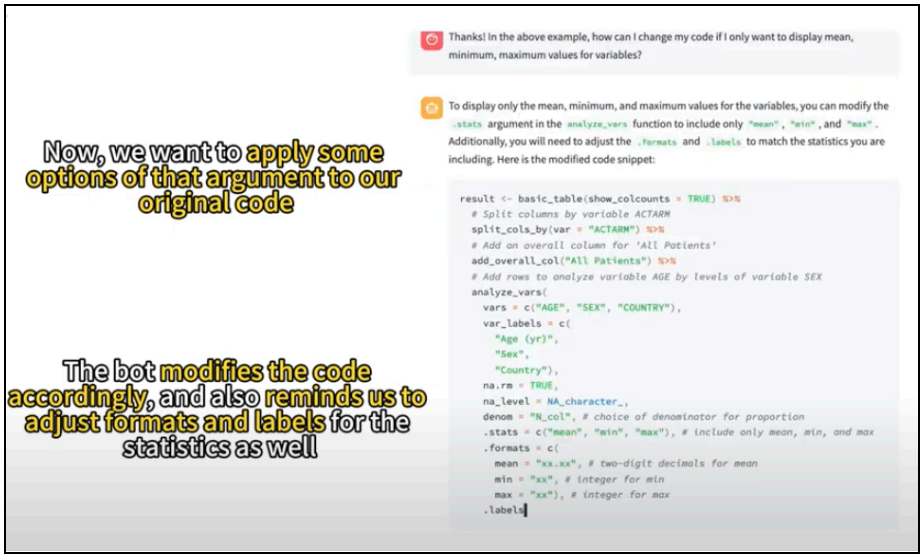


Roche also developed NExpert, a PoC chatbot utilizing the Azure OpenAI Studio GPT-4 API, to assist the code generation. It can 1) answer user's questions about functions & arguments in NEST packages, 2) suggest and modify output template codes via natural language prompts. Similar to the PharmaVerse Coding Assistant GPT, it stored the documents at the back end, but it has more freedom on pre-defining the documentation searching strategy. Depending on a user's question, NExpert may go through a different searching path to find the information for answer generation, which will potentially decrease the waiting time and increase the quality of answers. Below are several screenshots showing NExpert completing tasks based on natural language prompts.




We then ask the bot how to display columns using variable ARM
The bot suggests modifications on existing code
It also reminds us to use the correct variable name
mean_sd = "Mean (SD)",
median = 'Median')
) %>%
build_table(adsl)
result
This code will create a table with columns split by the treatment arm (ACTARM) and an additional column for all patients. It will show the number of observations, mean and standard deviation, median, range, and count fraction for Age, and the count fraction for Sex and Country. Adjust the variable names and labels as needed to match your dataset.
Using this example, how can I use ARM instead of actual arm to display columns?
To use "ARM" instead of "ACTARM" for column splitting, you can modify the split_cols_by function argument from var = "ACTARM" to var = "ARM" in your code. Here is the modified code snippet:
result <- basic_table(show_colcounts = TRUE) %>%
# Split columns by variable ARM
split_cols_by(var = "ARM") %>%
... (rest of the code remains unchanged)
Make sure to replace "ARM" with the exact column name in your dataset that represents the treatment arm if it is different.


We now ask the bot to retrieve additional information on a function argument
The bot displays all the available information from package documentation and explains how to use the argument
Cool! I'd like to know if there are other statistics I can use to summarize these variables. Can you show me the available stats options?
Certainly! The available statistics options for summarizing variables in the rtables and tern packages are quite extensive. Here's a list of statistics you can use with the s_summary function for numeric variables:
n : The number of non-NA values.
sum : The sum of values.
mean : The mean of values.
sd : The standard deviation.
se : The standard error of the mean.
mean_sd : The mean and standard deviation.
mean_se : The mean and standard error.
mean_ci : The confidence interval for the mean.
mean_sei : The standard error interval for the mean.
mean_sdi : The standard deviation interval for the mean.
mean_pval : The two-sided p-value for the mean.
median : The median of values.
mad : The median absolute deviation.
median_ci : The confidence interval for the median.
quantiles : Two sample quantiles.
iqr : The interquartile range.
range : The range of


Now, we want to apply some options of that argument to our original code
The bot modifies the code accordingly, and also reminds us to adjust formats and labels for the statistics as well
Thanks! In the above example, how can I change my code if I only want to display mean, minimum, maximum values for variables?
To display only the mean, minimum, and maximum values for the variables, you can modify the .stats argument in the analyze_vars function to include only "mean" , "min" , and "max" . Additionally, you will need to adjust the .formats and .labels to match the statistics you are including. Here is the modified code snippet:
result <- basic_table(show_colcounts = TRUE) %>%
# Split columns by variable ACTARM
split_cols_by(var = "ACTARM") %>%
Add an overall column for 'All Patients'
add_overall_col("All Patients") %>%
Add rows to analyze variable AGE by levels of variable SEX
analyze_vars(
vars = c("AGE", "SEX", "COUNTRY"),
var_labels = c(
"Age (yr)",
"Sex",
"Country"),
na.rm = TRUE,
na_level = NA_character_,
denom = "N_col", # choice of denominator for proportion
.stats = c("mean", "min", "max"), # include only mean, min, and max
.formats = c(
mean = "xx.xx", # two-digit decimals for mean
min = "xx", # integer for min
max = "xx"), # integer for max
.labels

These use-cases show promising prospects for LLMs as an assistant for the future output generation, which will significantly reduce the resource required and lower the barrier to data and coding knowledge.

Agents & Action Tools

Programming often includes productionalizing code and workflows. This process is regularly referred to as continuous integration and continuous delivery/continuous deployment (CI/CD). Popular CI/CD tools include GitHub Actions and Jenkins. Programmers automate workflows with CI/CD focused on tasks or actions to complete. For example, a Shiny programmer may first require a test to pass before kicking off the steps of deploying an application to a production environment. An example of this is here:
https://github.com/colearendt/shinytest-example/blob/main/.github/workflows/run-shinytest.yaml

Organizations are starting to take this type of process and deployment further with AI. Organizations are configuring and deploying Autonomous AI agents which embark on a task (scraping sites, etc.) or goal by breaking down tasks to execute and reiterate on results. These agents are often purpose-built for that particular task. An open-source example is AgentGPT:
https://github.com/reworkd/AgentGPT

AgentGPT comes pre-configured with a CLI that includes various technical components like a frontend, AI backend, and database. Agents often require connections to various software environments (APIs, etc.) and can be costly given the repeated API calls. Another popular agent is AutoGPT:
https://github.com/Significant-Gravitas/AutoGPT

Many large tech companies (Google AI Studio, Azure OpenAI etc.) are providing Agents and active programming tools as well as groups like Anthropic, HuggingFace, and Perplexity etc. Users of ChatGPT and others like Perplexity tend to focus and benefit from open-ended searches about programming. The focus often with many of these agents is on code development and review. These activities help in creating new code as well as improving existing workflows. This includes GPT-4 *advanced data analysis* (previously Code Interpreter) from OpenAI. While marketed as a plugin, *advanced data analysis* can execute code (in various programming languages) for ChatGPT Plus users and generate output. *In addition to providing code, advanced data analysis* can add explanations and analyze data. You can read more about it here:
https://help.openai.com/en/articles/8437071-advanced-data-analysis-chatgpt-enterprise-version

Microsoft has an open source tool also as seen below:
https://www.microsoft.com/en-us/research/project/autogen/overview/

One can imagine a future where tools will have the required approvals to further enhance an analysis while also incorporating production activities, deployments and other activities. Some organizations are creating internal AI agents or offering agents for sale. Shopify for example has various AI Agents available for Shopify stores. These agents can answer customer questions, resolve inquiries using automation and even take action like applying discounts on store items based on instructions.

The focus on agents, actions and tasks has led to discussions around Large Action & Behavioral Models (LAM). While it is unclear how LAMs differ from LLMs, it does seem the key focus is on taking action such as analyzing data and then emailing the results. Much is yet to be seen regarding how these actions will be managed and authorized for statistical programmers in pharma given sensitive data, security and privacy efforts. Managed cloud services and lake houses could provide integrated support for actions on private data and systems further driving AI action oriented requests and tasks. You can read more here:
https://www.microsoft.com/en-us/research/project/ai-frontiers-explorations/
https://hbr.org/2023/11/how-companies-can-build-trustworthy-ai-assistants

Conclusion

This paper was to highlight how statistical programmers can use GenAI for statistical programming. As interest in AI increases by clinical statistical programmers, it is important to understand how this tooling can be used. We discussed tools programmers can use today as well as evolving topics. The information outlined in this paper highlights the evolving space of AI and use cases on how it can be used today.

CONTACT INFORMATION

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

Phil Bowsher
phil@posit.co