

Shiny & LLMs: Landscape and Applications in Pharma

Phil Bowsher, Posit; Boston, USA

Abstract

Shiny is impactful in the pharmaceutical industry as it helps people share statistical programming and analysis via a web-based interface. The interactive web-based capabilities help users explore analysis and see results quicker than in static output. Shiny is applied in drug discovery to submissions and beyond as highlighted in the article:

<https://posit.co/blog/shiny-use-cases-within-pharma/>

Pharmaceutical organizations have built an exploration framework on top of Shiny for exploratory data analysis via teal: <https://insightsengineering.github.io/teal/latest-tag/>

The R in Pharma conference each year features talks and workshops on Shiny which can be seen here:

<https://www.youtube.com/c/RinPharma>

There are various use-cases for GenAI to further the power Shiny in pharma. Many of these examples were highlighted at the GenAI in Pharma event in 2024 found here:

<https://www.youtube.com/playlist?list=PLMtxz1fUYA5AYryl4t2mtqBngqWDmMXJ>

This paper will provide detail on the evolving nature of Shiny to support using generative artificial intelligence (GenAI) with an emphasis on clinical and statistical programming tasks.

Introduction

Many pharmaceutical organizations are migrating from commercial software to open source in drug development. Pharmaceutical companies like Roche are experiencing various benefits of large language models (LLMs). Roche recently provided the details about its GenAI Assistant for Clinical Programming team: *"We have over 1000 users with 44000 questions asked and answered already, specifically, 4300 production runs on OCEAN have been debugged by OCEAN Assistant. Putting things into perspective, we believe it takes an average of at least 15 minutes to get any question answered by one data scientist alone, including time spent on documentation search and also reaching out to SMEs. Given that OCEAN Assistant gives the answer instantly with a great level of accuracy and details, we believe that we could save about 25 hours per 100 questions asked on OCEAN Assistant."*

https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PAP_OS01.pdf

Moreover, teams like Servier created a Shiny app called "Spot it" for Visual QC at scale. Servier highlighted in a recent talk that the app is "Future-ready with planned AI enhancements" as seen here:

https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PRE_SM04.pdf

Other Pharmaceutical companies are integrating GenAI into Shiny development as in *"AI-assisted test generation offers a solution by leveraging large language models (LLMs) to automatically generate {testthat} unit tests and {shinytest2} snapshot tests, significantly reducing the manual effort required for test creation"* which was highlighted by Emily Yates at Formation Bio here:

https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PAP_SM11.pdf

Below will provide details of examples how Pharmaceutical companies can utilize LLMs with Shiny with an emphasis on using R and python for clinical use-cases.

Ecosystem of LLMs: Evolving Quickly

This paper will highlight current tools for combining Shiny and GenAI. This ecosystem is evolving quickly and changes feel to be happening weekly. For current information on GenAI tools from Posit PBC, please consult: <https://posit.co/use-cases/ai/>

When discussing GenAI, there are 3 layers that often come up:

1. Users Programming Experience and Skills (Beginner, Intermediate and Advanced etc.)
2. GenAI Applications (Code Completions, Chat-based Programming, Chatbots, Vibe coding, Agents/Action based etc.)
3. Industry Applications (Interface development, Training users, Report generation, Clinical analysis etc.)

By dissecting these layers in conversations, it often helps reveal the tool/application best suited for the clinical use-case.

Posit Tools as April 2025, for Large Language Models (LLM) in R & Python

While there are many LLM based tools for R and python users, below will highlight 2 key packages from Posit PBC:

1. *ellmer* for R

ellmer makes it easy to use LLMs from R. It supports many LLM providers and offers rich capabilities to use within various integrated development environments like RStudio, Positron and VS Code. The *ellmer* main page is: <https://github.com/tidyverse/ellmer>

2. *chatlas* for Python

Like *ellmer*, *chatlas* provides an interface to LLM providers, but for Python users. The *chatlas* main page is: <https://posit-dev.github.io/chatlas/>

Below will detail how to use *ellmer* and *chatlas* for creating Shiny apps. Another tool to know about is Github Copilot. You can see an explanation of creating Shiny apps with Github Copilot here:

GenAI and Pharma: Learning as we go | Episode 1: Copilot

https://www.youtube.com/watch?v=fnfVCMsXTy0&ab_channel=PositPBC

Using *ellmer* & *chatlas*

Both *ellmer* and *chatlas* can be used in various ways by programmers as detailed below. First, users will specify the provider/model choice. This usually entails obtaining an API key from the model organization and then store that securely in your environment. Posit PBC recommends saving an environment variable rather than using the API key directly in your code. After securely connecting to your LLM of choice (local or external), each connection will start by creating a new chat object like this in R:

```
library(ellmer)

chat <- chat_openai(
  model = "gpt-4o-mini",
  system_prompt = "You are a friendly but terse assistant.",
)
```

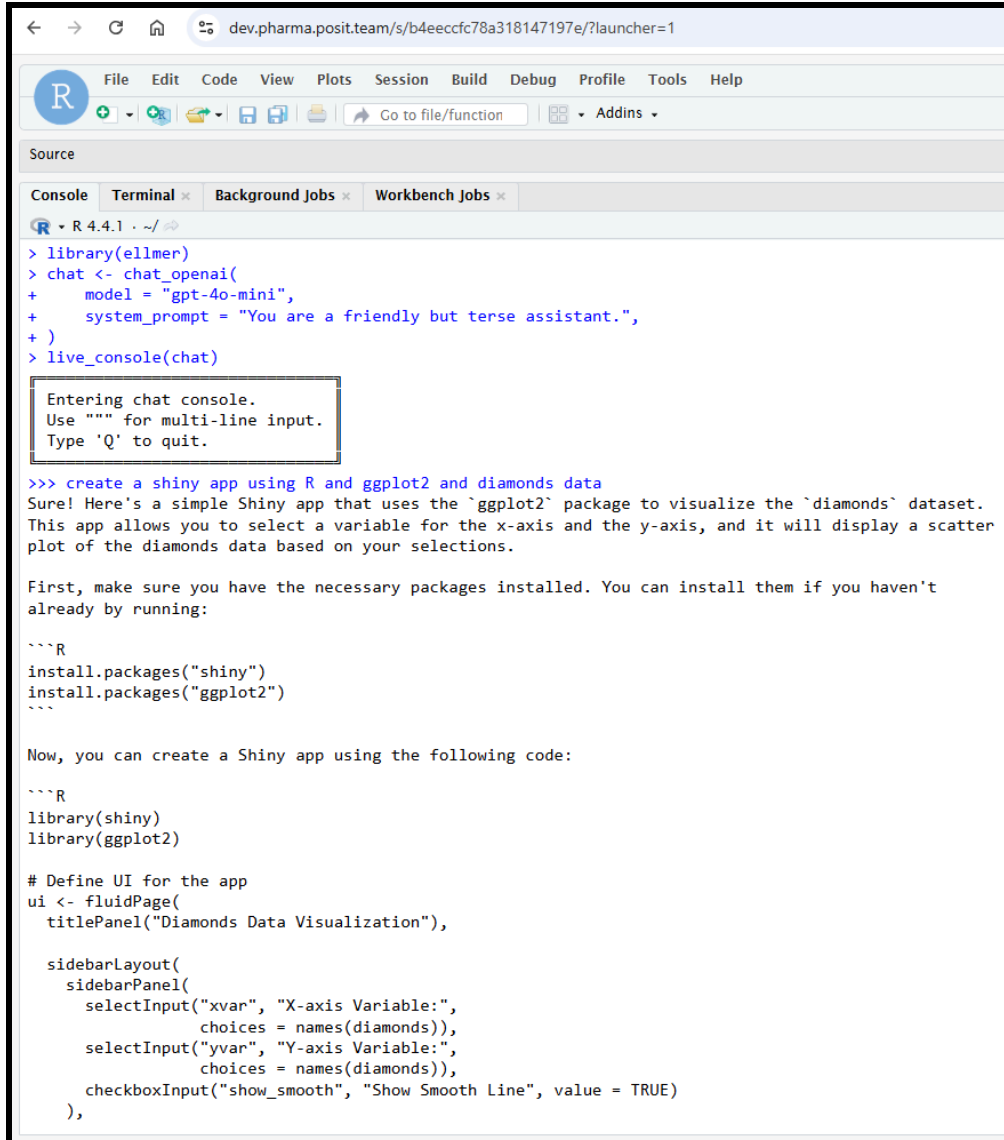
This code can be executed in the R console, in a R file or Quarto document, using an IDE of choice like RStudio or Positron. Below are the various ways programmers can use *ellmer* and *chatlas*:

A. Interactive chat console

This is the most interactive and least programmatic method of using ellmer, by chatting directly in your R console or browser with `live_console(chat)` or `live_browser()` like:

```
live_console(chat)
```

This will create a dedicated prompting session in RStudio or Positron like:



The screenshot shows an RStudio interface with the following content in the console:

```
R > R 4.4.1 ~ /  
> library(ellmer)  
> chat <- chat_openai(  
+   model = "gpt-4o-mini",  
+   system_prompt = "You are a friendly but terse assistant.",  
+ )  
> live_console(chat)
```

A text box appears with the message: "Entering chat console. Use "" for multi-line input. Type 'Q' to quit."

```
>>> create a shiny app using R and ggplot2 and diamonds data  
Sure! Here's a simple Shiny app that uses the `ggplot2` package to visualize the `diamonds` dataset. This app allows you to select a variable for the x-axis and the y-axis, and it will display a scatter plot of the diamonds data based on your selections.  
  
First, make sure you have the necessary packages installed. You can install them if you haven't already by running:  
  
```R  
install.packages("shiny")
install.packages("ggplot2")
```  
  
Now, you can create a Shiny app using the following code:  
  
```R  
library(shiny)
library(ggplot2)

Define UI for the app
ui <- fluidPage(
 titlePanel("Diamonds Data Visualization"),

 sidebarLayout(
 sidebarPanel(
 selectInput("xvar", "X-axis Variable:",
 choices = names(diamonds)),
 selectInput("yvar", "Y-axis Variable:",
 choices = names(diamonds)),
 checkboxInput("show_smooth", "Show Smooth Line", value = TRUE)
),
 mainPanel(
 plotOutput("p1")
)
)
)
server <- function(input, output, session) {
 output$p1 <- ggplot(diamonds, aes(
 x = get_xvar(),
 y = get_yvar()
))
 if (input$show_smooth) {
 output$p1 <- output$p1 + geom_smooth()
 }
 output
```

### B. Interactive method call

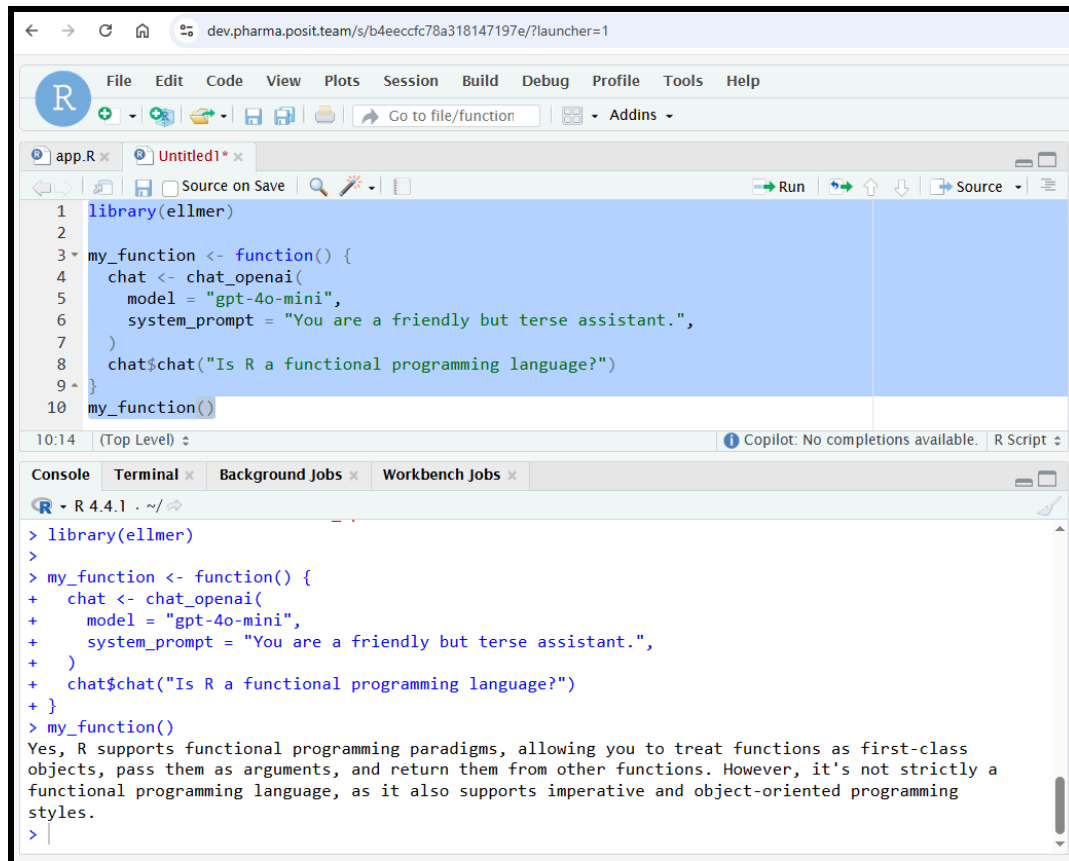
Another interactive way to chat is to call the `chat()` method. This can be run directly in your R code as:

```
result <- chat$chat("Tell me a story")
result
```

### C. Programmatic chat

Also, users can create the chat object inside a function, which can then be called programmatically with code such as `my_function()`. Below is an example of using `ellmer` with programmatic chat:

```
my_function <- function() {
 chat <- chat_openai(
 model = "gpt-4o-mini",
 system_prompt = "You are a friendly but terse assistant.",
)
 chat$chat("Is R a functional programming language?")
}
my_function()
```



```
1 library(ellmer)
2
3 my_function <- function() {
4 chat <- chat_openai(
5 model = "gpt-4o-mini",
6 system_prompt = "You are a friendly but terse assistant.",
7)
8 chat$chat("Is R a functional programming language?")
9 }
10 my_function()

10:14 (Top Level) Copilot: No completions available. R Script

Console
R - R 4.4.1 - ~/...
> library(ellmer)
>
> my_function <- function() {
+ chat <- chat_openai(
+ model = "gpt-4o-mini",
+ system_prompt = "You are a friendly but terse assistant.",
+)
+ chat$chat("Is R a functional programming language?")
+ }
> my_function()
Yes, R supports functional programming paradigms, allowing you to treat functions as first-class
objects, pass them as arguments, and return them from other functions. However, it's not strictly a
functional programming language, as it also supports imperative and object-oriented programming
styles.
>
```

## Use-cases for Shiny and GenAI

Below will discuss applications of Shiny and GenAI utilizing tools like `ellmer` and `chatlas`. While the examples below are featured in the R and Python package Shiny, other application packages can be used, like Streamlit, with `ellmer` and `chatlas` as detailed here: <https://posit-dev.github.io/chatlas/web-apps.html>

### 1. Chatbots via Shiny

#### 1. shinychat for Shiny for R

`shinychat` is a Chat UI component for Shiny for R. It is implemented via the R package `shinychat` and it is designed to work with `ellmer` and other chat clients. As detailed above, `ellmer` can work with various LLMs, either local or vendor provided. `shinychat` works by creating a stream that contains the user input. The main page can be found at:

<https://posit-dev.github.io/shinychat/>

Below is some sample code to create a shinychat application in R:

```
library(shiny)
library(shinychat)

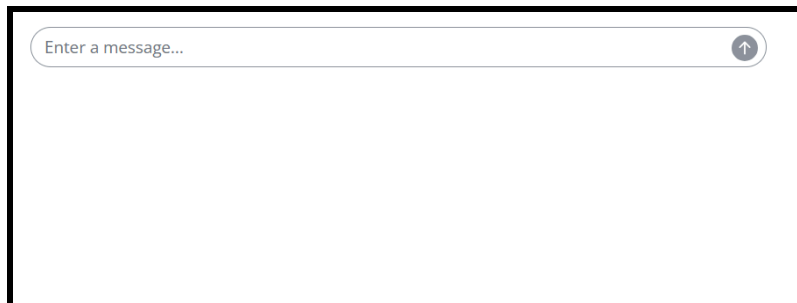
ui <- bslib::page_fluid(
 chat_ui("chat")
)

server <- function(input, output, session) {
 chat <- ellmer::chat_openai(system_prompt = "You're a trickster who answers in riddles")

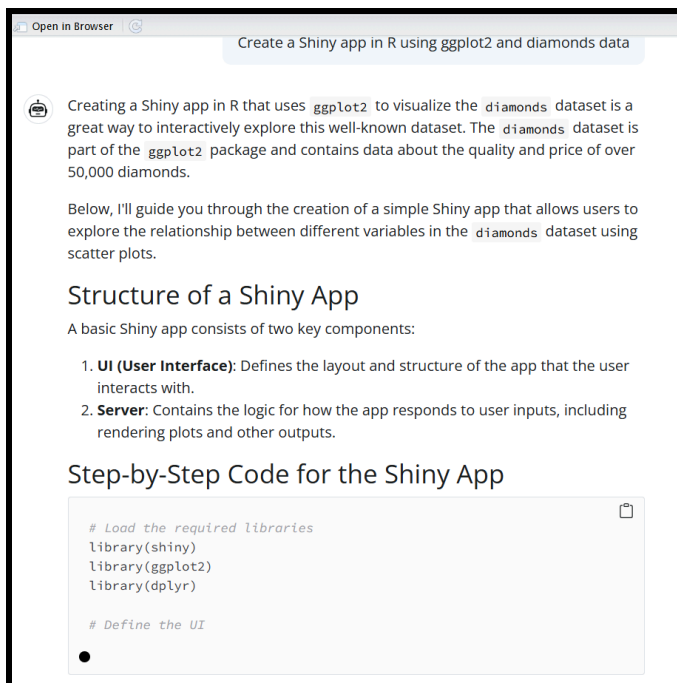
 observeEvent(input$chat_user_input, {
 stream <- chat$stream_async(input$chat_user_input)
 chat_append("chat", stream)
 })
}

shinyApp(ui, server)
```

After the code runs, it will create a Shiny app like:



Users then can chat with the LLM as in:



Open in Browser

### Create a Shiny app in R using ggplot2 and diamonds data

Creating a Shiny app in R that uses `ggplot2` to visualize the `diamonds` dataset is a great way to interactively explore this well-known dataset. The `diamonds` dataset is part of the `ggplot2` package and contains data about the quality and price of over 50,000 diamonds.

Below, I'll guide you through the creation of a simple Shiny app that allows users to explore the relationship between different variables in the `diamonds` dataset using scatter plots.

## Structure of a Shiny App

A basic Shiny app consists of two key components:

1. **UI (User Interface):** Defines the layout and structure of the app that the user interacts with.
2. **Server:** Contains the logic for how the app responds to user inputs, including rendering plots and other outputs.

## Step-by-Step Code for the Shiny App

```
Load the required libraries
library(shiny)
library(ggplot2)
library(dplyr)

Define the UI
```

A live example of *shinychat* can be seen and used with *ellmer*-assistant: <https://github.com/jcheng5/ellmer-assistant>

Users can explore and try the application here: <https://jcheng.shinyapps.io/ellmer-assistant/>

It is created by adding the README.md files from *ellmer* and *shinychat* into a system prompt for GPT-4o as seen here: <https://github.com/jcheng5/ellmer-assistant/blob/main/prompt.generated.md>

Another example can be found by Kamil Wais at Roche. Kamil created a package called *GitAI*, that extracts knowledge from 'Git' repositories. Kamil created a Shiny app as a chatbot where answers are based on the results from processed content of multiple git repositories as seen here: <https://kalimu.shinyapps.io/GitAI-demo/>

The application is built with many R packages, including *shinychat* as discussed above. You can read more about the application here: <https://cran.rstudio.com/web/packages/GitAI/vignettes/building-shiny-app-chatbot.html>

## 2. Chatbot for Shiny for Python

To build a *Shiny for Python* chatbot, we will utilize the `ui.Chat` component, where users pass user input from the component into the `chat.stream()` method, similar to *Shiny for R*. Below will use *Shiny for Python* and *Anthropic* to create a chatbot. First, run the following code in an `app.py` file:

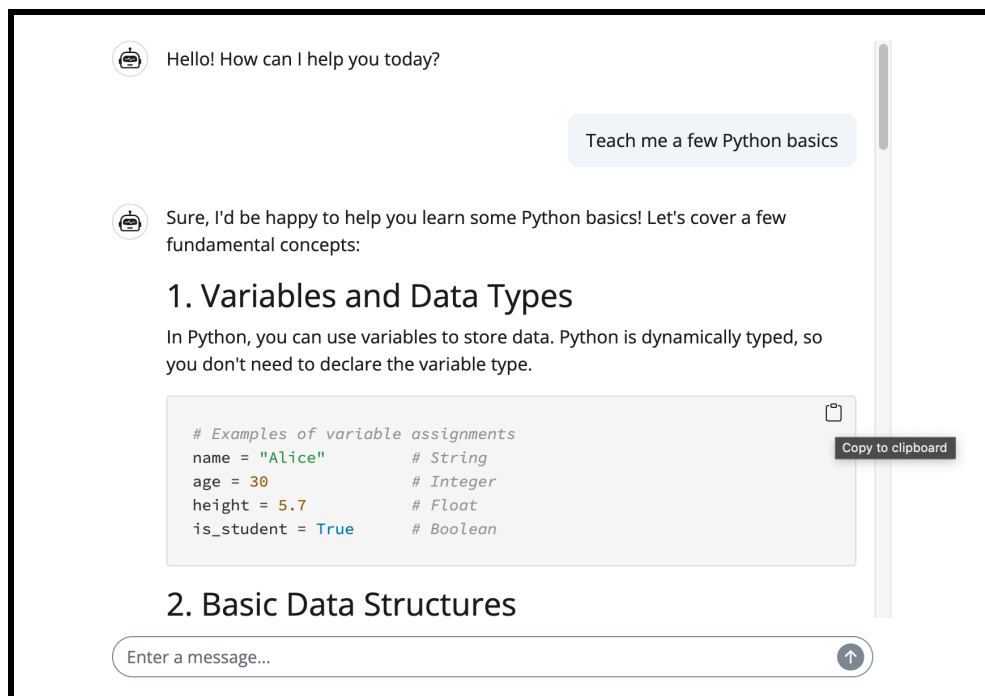
```
from chatlas import ChatAnthropic
from shiny.express import ui

chat = ui.Chat(
 id="ui_chat",
 messages=["Hi! How can I help you today?"],
)
chat.ui()

chat_model = ChatAnthropic()

@chat.on_user_submit
async def handle_user_input():
 response = chat_model.stream(chat.user_input())
 await chat.append_message_stream(response)
```

After the code runs, it will generate the *Shiny for Python* app like:



We will further explore *Shiny for Python* for creating custom LLM Shiny interfaces in Python later in this paper. You can read more about creating chatbots for *Shiny for Python* here:

<https://shiny.posit.co/py/docs/genai-chatbots.html>

One could imagine adding chatbot capabilities to teal applications for clinical data exploration as stated here “Integrate GenAI to further Pave the way to end-to-end Clinical Reporting automation” for the {teal.builder} tool: [https://phuse.s3.eu-central-1.amazonaws.com/Archive/2024/Connect/US/Bethesda/PRE\\_ET10.pdf](https://phuse.s3.eu-central-1.amazonaws.com/Archive/2024/Connect/US/Bethesda/PRE_ET10.pdf)

## 2. ***Sidebots via Shiny***

The above examples are Shiny interfaces which operate as a chatbot in R and Python. The application's primary goal is to provide a chatbot for users to prompt and chat into, while the output streams down the page. Below we will discuss the use-case where users would like to create an application that has a sidebot for helping to drive the data science within an application. Jazz Pharmaceuticals recently highlighted a need for “Interactive Patient Profiling applications to help clinicians visualize participant profiles, including placebo risk and dropout probability for more informed decisions”, where one could also see having GenAI prompting or support as an enhancement. Further information is here: [https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PRE\\_ML02.pdf](https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PRE_ML02.pdf)

### 1. ***querychat***

querychat is a multilingual package that allows you to chat with your data using natural language queries and available for *Shiny for R* as well as *Shiny for Python*. Querychat main page is found at:

<https://github.com/posit-dev/querychat>

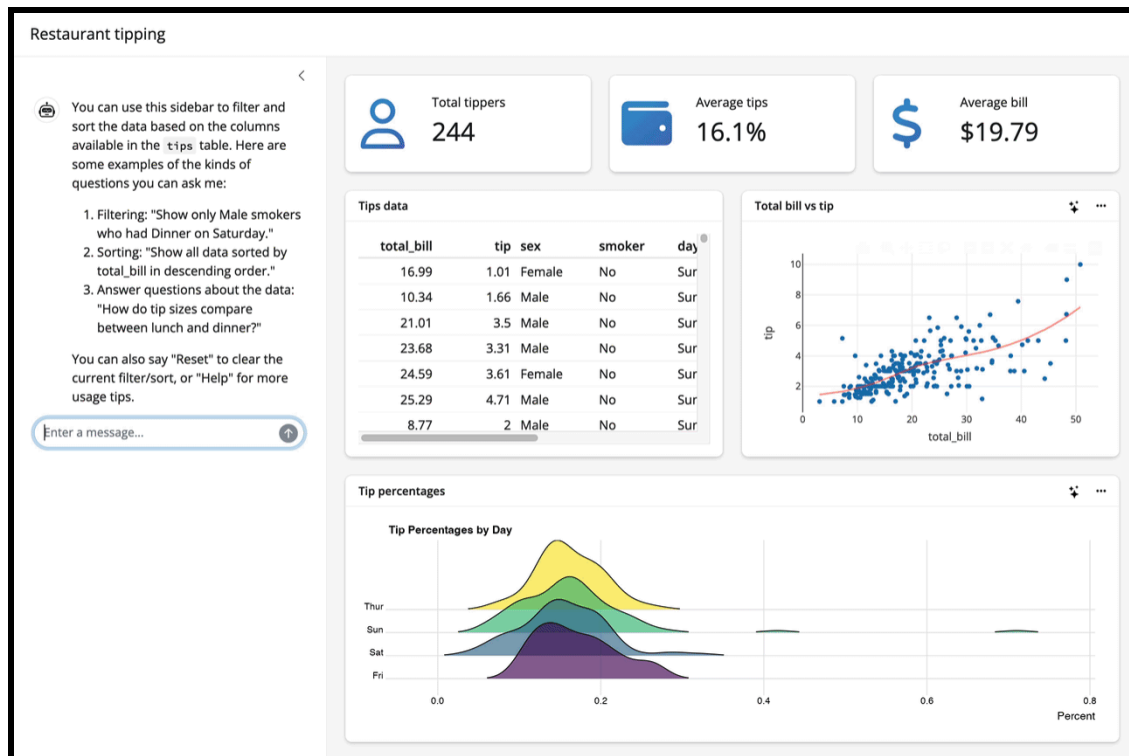
In R, querychat is installed via an R package via:

```
pak::pak("posit-dev/querychat/r-package")
```

In Python, querychat is installed via:

```
pip install "querychat @ git+https://github.com/posit-dev/querychat#subdirectory=python-package"
```

*querychat* is a drop-in component for Shiny that provides users in a Shiny app the ability to query a data frame using natural language. The analysis results are available in Shiny's reactive data frame, and therefore can be used from Shiny functions directly etc. *querychat* does not access the application directly, but rather is forced to write SQL queries. *querychat* uses DuckDB for its SQL engine. As with the tools above, *querychat* can be powered by LLMs like GPT-4o, Claude 3.5 Sonnet, etc. Below is an example of using *querychat*:



You can find examples of Shiny for R and Shiny for Python code here:

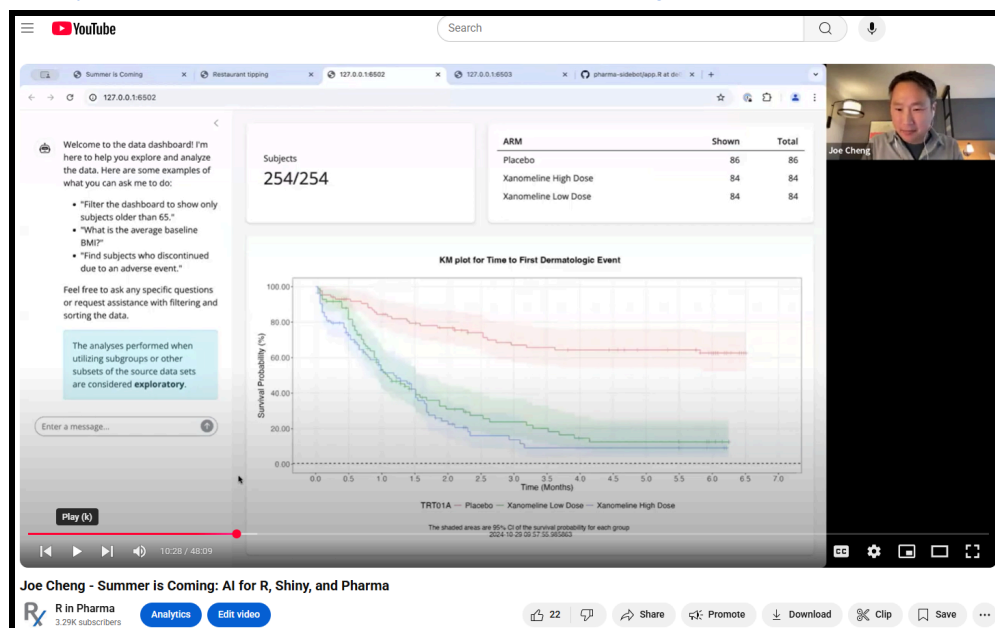
<https://github.com/jcheng5/r-sidebot>

<https://github.com/jcheng5/py-sidebot>

Inspecting the code above will highlight some similar packages, `ellmer` and `chatlas`!

Joe Cheng, the creator of Shiny, gave the Keynote at R/Pharma 2024 on creating a Shiny Pharma Sidebot. The Keynote presentation is available at:

[https://youtu.be/AfMa1CVUdXU?list=PLMTxz1fUYA5Bz0CQHqu4MTr-DWRVviT8\\_&t=621](https://youtu.be/AfMa1CVUdXU?list=PLMTxz1fUYA5Bz0CQHqu4MTr-DWRVviT8_&t=621)





The code for the application above is at: <https://github.com/jcheng5/pharma-sidebot>

The application is a demonstration of using an LLM to enhance a data dashboard written in Shiny and is based on code and data from the Simulated data from CDISC Pilot and the R Submissions Working Group Pilot 2 Project: <https://rconsortium.github.io/submissions-wg/pilot2.html>

An example of how to use the querychat to work on internal or custom data can be found here: <https://www.appsilicon.com/post/r-sidebot#your-data>

### 3. ***Databots via Shiny***

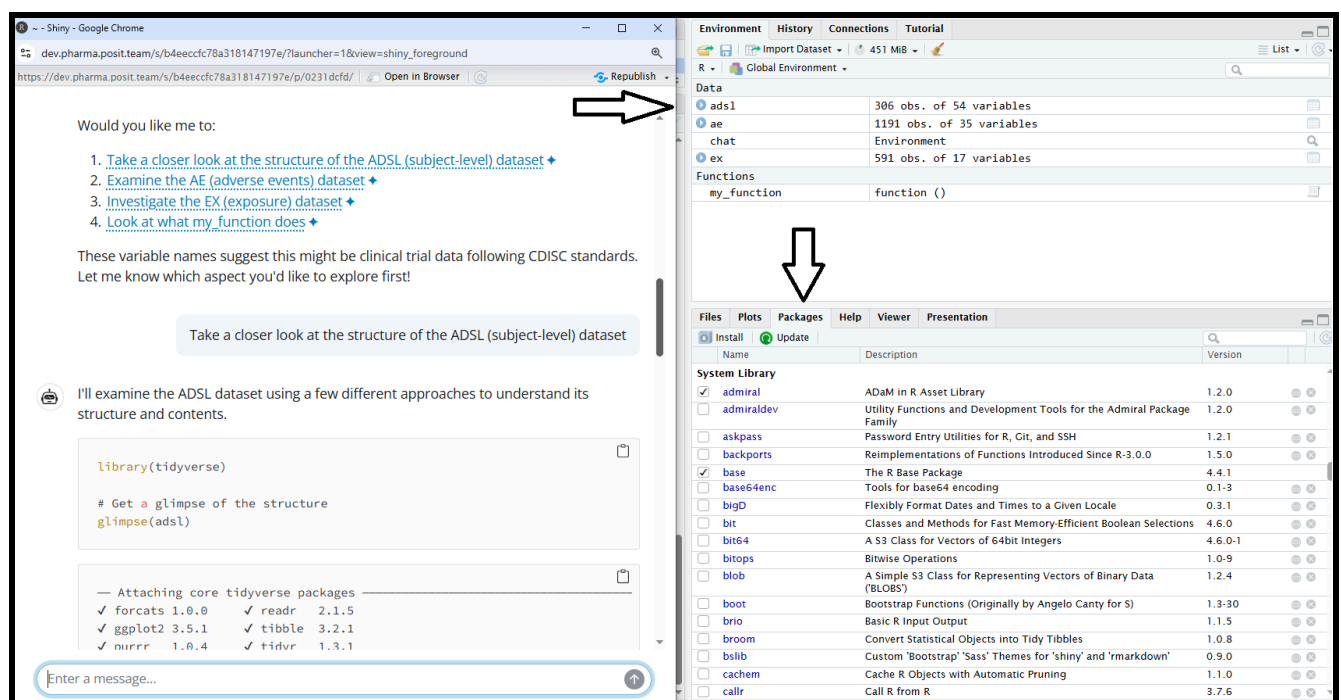
The above tools utilize *ellmer* and *chatlas* to help add chatting capabilities as well as analysis to Shiny applications (R or Python). Databot is an experimental AI assistant that is designed to help users complete exploratory data analysis by performing tasks within your R session (in RStudio, Positron or other IDE). The main page can be found at: <https://github.com/jcheng5/databot>

Databot uses Claude 3.5 Sonnet, and requires an ANTHROPIC\_API\_KEY environment variable. It will use your environment data as well as your installed packages. Below we will highlight this tool in RStudio and discuss future implications in Positron. Databot uses many of the same tools and packages above, but is instructed “to assist the user with data analysis, manipulation, and visualization tasks” as seen here: <https://github.com/jcheng5/databot/blob/main/inst/prompt/prompt.md>

To install Databot, run the following R code in the R console:

```
pak::pak("jcheng5/databot")
```

Then to run the Databot locally within the RStudio IDE, run the following **databot::chat()** to launch the app. This will open up the databot interface as shown below:



The databot is analysing the dataset, ADSL, from within the environment and also using packages local to the session. Users can use the Databot to specify certain areas of interest for analysis such as treatment distribution:

Would you like to explore any particular aspect of this dataset in more detail?

Suggested next steps:

1. [Let's look at the distribution of subjects across treatment arms](#) ♦
2. [Can we examine the age and gender distribution of the study population?](#) ♦
3. [Show me the reasons for study discontinuation \(EOSSTT\)](#) ♦
4. [Let's analyze the treatment duration \(TRTDURD\) across different arms](#) ♦

Let's look at the distribution of subjects across treatment arms

I'll help you analyze the distribution of subjects across treatment arms. Let's create a summary that shows both counts and percentages, and then visualize it.

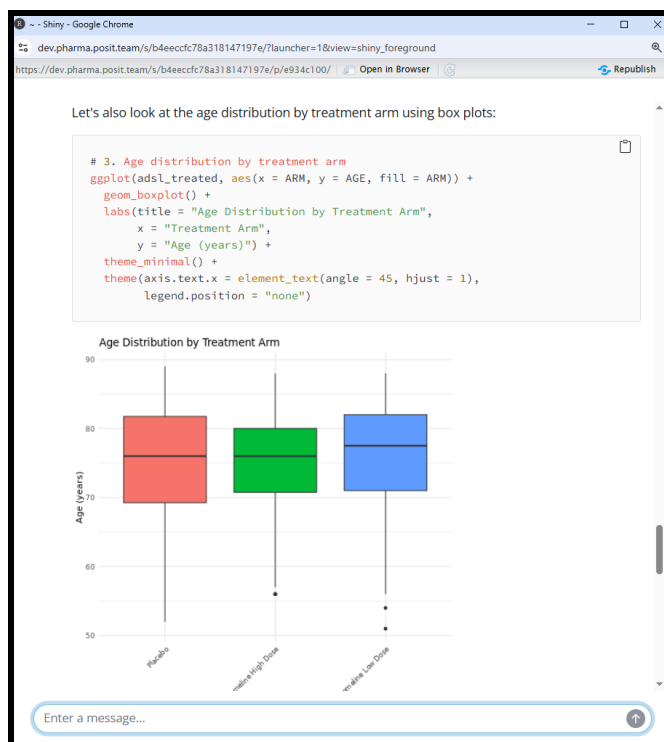
```
library(tidyverse)

Create a summary table
arm_summary <- adsl %>%
 count(ARM) %>%
 mutate(Percentage = round(n / sum(n) * 100, 1)) %>%
 arrange(desc(n))

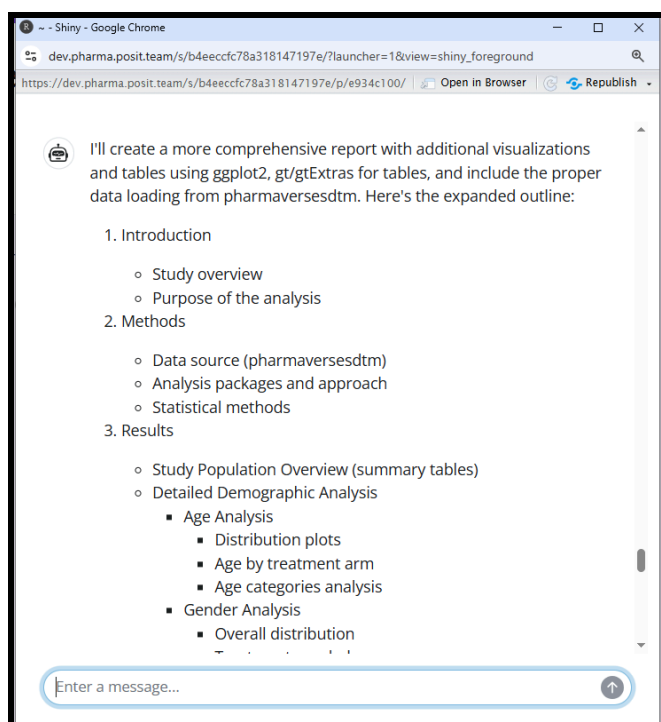
Display the summary table
```

Enter a message...

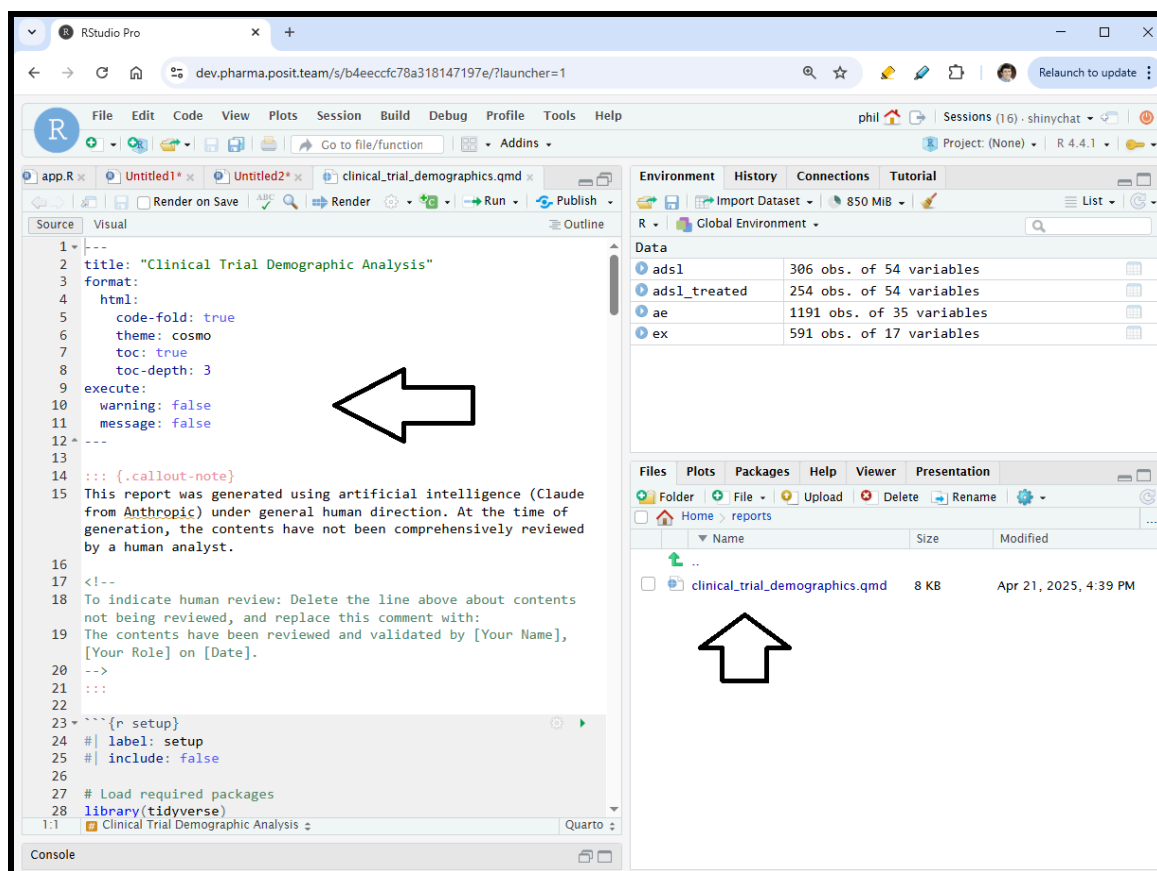
Users can also create data visualizations using Databot:



After analysis is finished, then Databot will help create a final report for review as seen below:



Here is the report in RStudio:

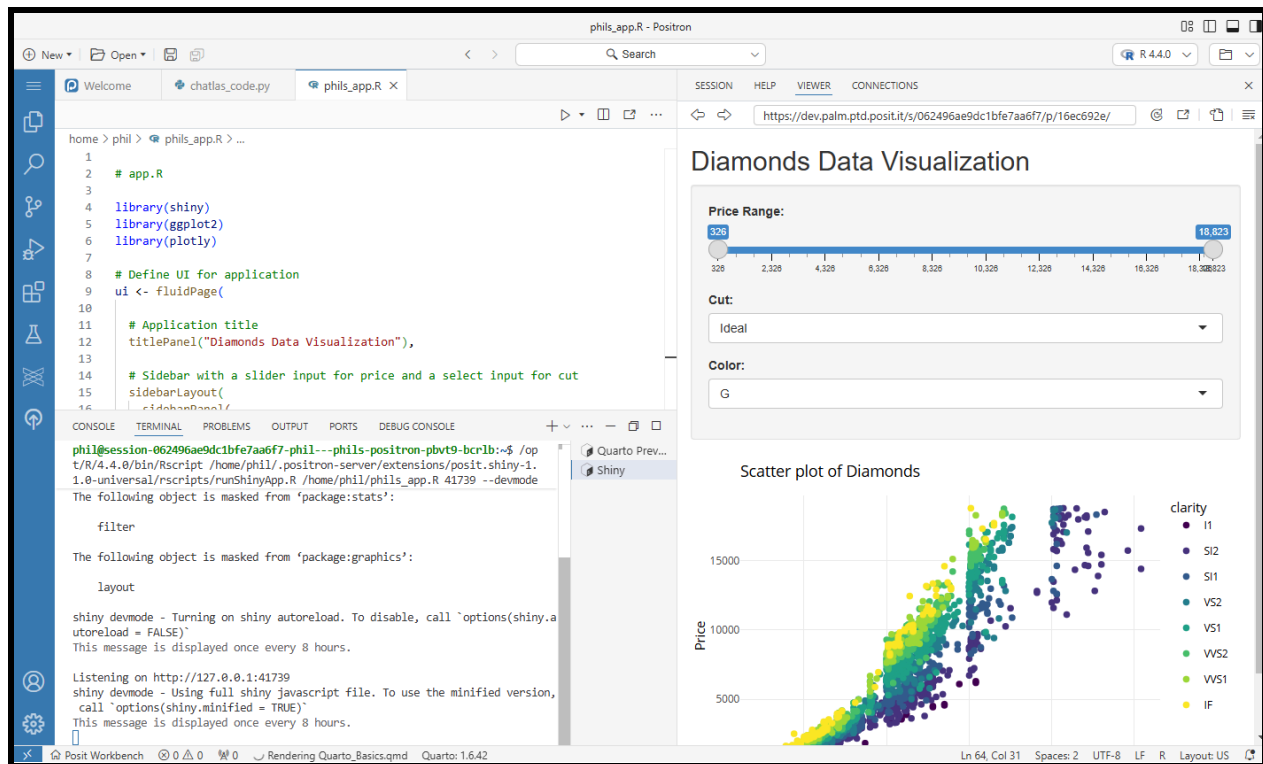


Users can either have Databot further update the report, or directly edit and run the code in an IDE like RStudio.

Python users can explore py-databot at the page: <https://github.com/icheng5/py-databot>

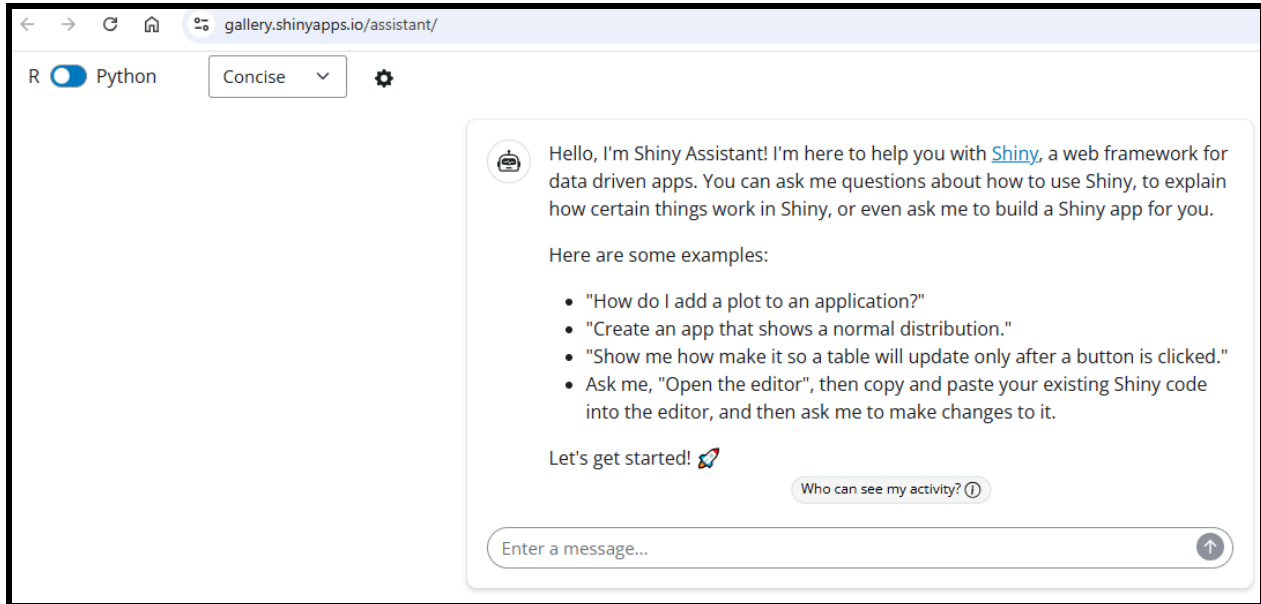
Extending Databot for specific analysis, like using the Pharmaverse, could be imaged by instructing and feeding the prompt various information on the Pharmaverse as in <https://pharmaverse.github.io/examples/> and package README files.

Positron, a new IDE from Posit PBC, was released in Summer of 2024. Below shows a Shiny app in Positron made with GenAI. All of the GenAI packages and tools above work in Positron. In addition to the packages above, the Positron Assistant will provide LLM integration with Positron, both for chat and for inline completions. It fulfills a role similar to Github Copilot. Announcements for this integration will be available here: <https://positron.posit.co/>

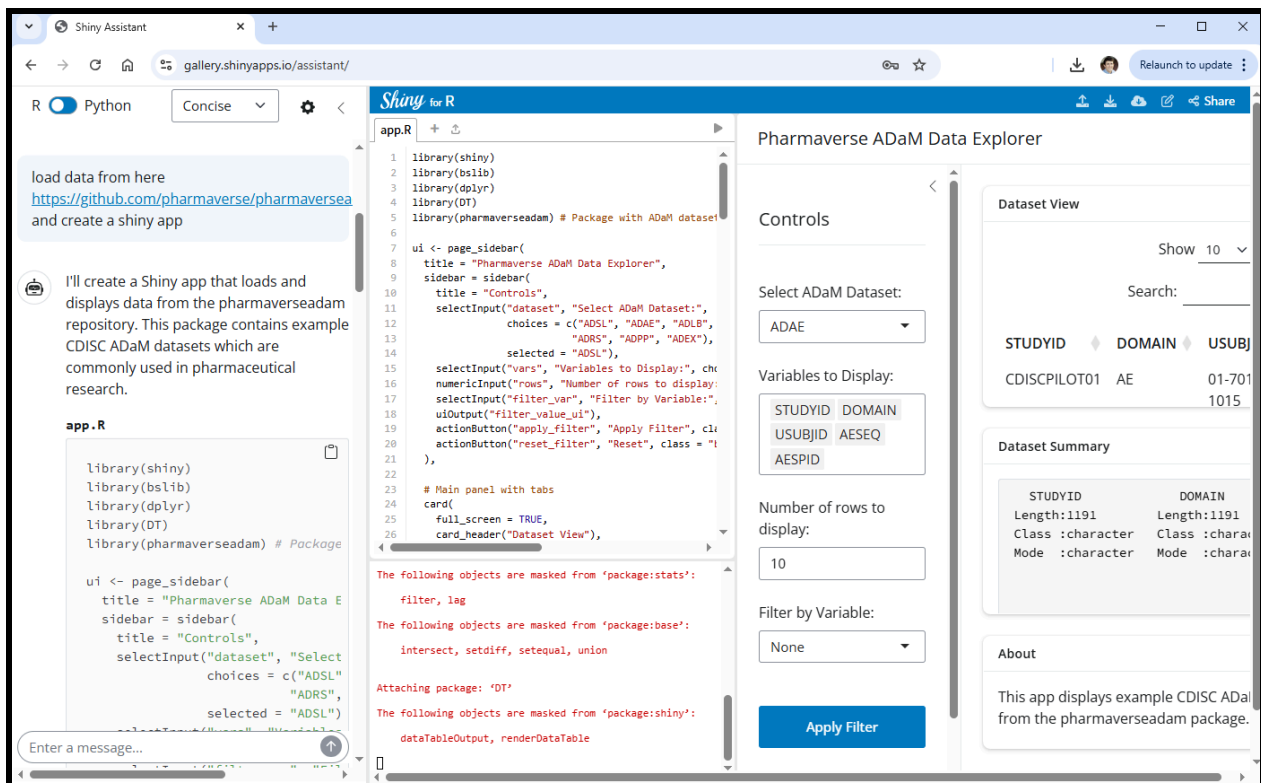


#### 4. **Shiny Assistant**

Shiny Assistant is an ai-powered tool that helps users create Shiny apps by promoting and asking questions about Shiny. Users can create new Shiny applications or ask questions and iterate on existing applications. Shiny Assistant acts as a knowledgeable guide to help people create Shiny apps in both R and in Python. People can test it out at: <https://gallery.shinyapps.io/assistant/>



Users can ask Shiny Assistant to create a Shiny app based on various conditions. Below is an example of creating a Shiny app that uses ADaM data from the *pharmaverseadam* package and creates an interactive ADaM application:



Users can interact with Shiny Assistant to make changes, and the original code will be used for edits and further updates. For an error, users can use the chat to ask the assistant to fix mistakes. Shiny Assistant builds applications using the Shinylive web interface. Shinylive is a special build of Shiny which runs R or Python (compiled to WebAssembly) in the web browser, which does not require a server to run R/Python apps. This is because Shinylive bundles a web server that can serve up static files. Not all packages can run in the browser and the code or data is shared with the user.

To further learn how to make Shiny apps for pharma using Shiny Assistant, please see:  
<https://posit.co/blog/ai-powered-shiny-app-prototyping/>

## **5. Shiny for Python for Custom LLM Interfaces**

While we believe that software development and data science happens in an IDE, there are many valid reasons to host an interface for non-coding professionals. Many pharmaceutical companies have robust and internal GenAI infrastructure and services such as Claude API, Amazon Bedrock, Azure OpenAI Studio GPT-4o API etc. These environments provide capabilities to use GenAI (Local, In-house or Foundational Models) privately and securely within an organization, extending a cloud vendor's resources into the organization's virtual private cloud (VPC). These services provide access to techniques such as fine-tuning and Retrieval Augmented Generation (RAG). Often organizations look to create interfaces into such tools within the intranet and firewall for non-coding professionals. An example of a custom interface into an GenAI environment is seen in the talk slides by Seamus Gallivan of Jazz Pharmaceuticals here:

[https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PRE\\_ML14.pdf](https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PRE_ML14.pdf)

While there are many UI options (Streamlit etc.), Shiny is a popular choice. If an organization has an internal local model (LLM) or API (plumber), *Shiny for Python* can be used to share it with non-programming professionals via an OpenAI type of interface. There are various Shiny options for surfacing the local LLM such as Shiny via R, Shiny with Reticulate, or Shiny for Python. Below we will discuss Shiny for Python. Further resources are available at:

<https://shiny.posit.co/blog/posts/shiny-python-1.0/>

<https://shiny.posit.co/py/components/display-messages/chat/#ai-quick-start>

To get started with Shiny for Python, users can view the template at:

<https://shiny.posit.co/py/docs/genai-chatbots.html>

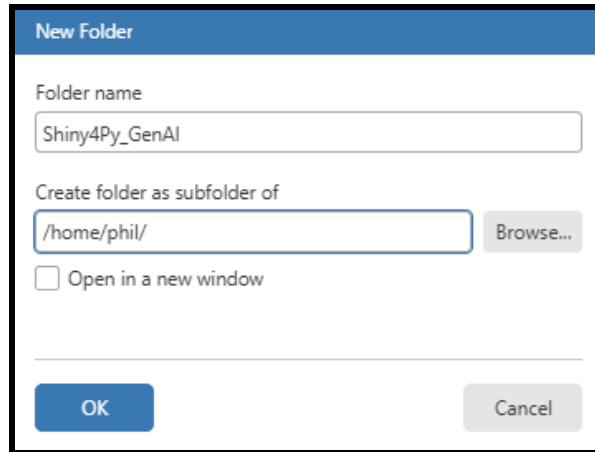
Some pharmaceutical companies create Shiny apps in R and Python using Reticulate. One such example is Kraken, a Shiny Application for Visualizing SDTM Data in Early Phase Clinical Trials by Servier. The team at Servier has also highlighted that *"Incorporating AI into Kraken offers significant potential to streamline the process of mounting new studies, reducing the time and effort needed to rewrite code for each analysis. By leveraging LLM's, Kraken could automatically adapt its workflows and visualizations to new datasets by recognizing patterns and mapping variables within the uploaded SDTM datasets. This capability would allow the system to intelligently configure plots, tables, and filters specific to each study's structure and focus area, minimizing manual intervention."* as found here:

[https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PAP\\_SM15.pdf](https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PAP_SM15.pdf)

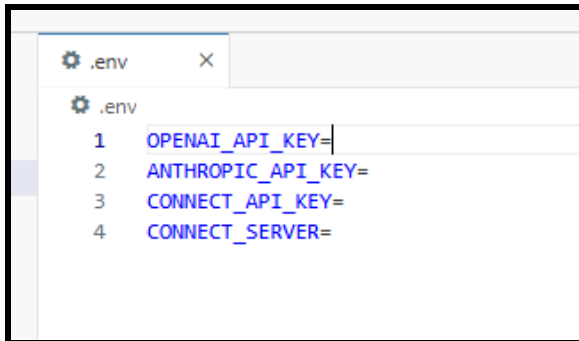
Below is an example of creating a Shiny for Python application as an interface to an LLM (local or via providers). The `ui.Chat()` component that is now bundled with *Shiny for Python* and will be used for our application. The new `Chat()` component makes it easy to implement generative AI chatbots, powered by any LLM of your choosing. The example below will use *chatlas* to interface with the LLMs.

The main steps of creating a GenAI chatbot with Shiny for Python in Positron IDE are:

1. Create a new folder in Positron. Example below is called *Shiny4Py\_GenAI*:



- When using chatlas, it is recommended to use environment variables or a configuration file to manage credentials. A popular method to manage credentials is to use a `.env` file to store your credentials. To create a new `.env` file in Positron, type "New File" and enter `.env`. This file needs to be stored in the App folder. In the file, add the following:



- Depending on your LLM provider of choice, you will need to run the following template installation in the Terminal in Positron. Other LLM provided templates can be found here: <https://shiny.posit.co/py/docs/genai-chatbots.html>  
Below is an example for Anthropic:

```
shiny create --template chat-ai-anthropic
```

Below is the code for the application:

```
import os

from app_utils import load_dotenv
from chatlas import ChatAnthropic

from shiny.express import ui

load_dotenv()
chat_client = ChatAnthropic(
 api_key=os.environ.get("ANTHROPIC_API_KEY"),
 model="claude-3-7-sonnet-latest",
 system_prompt="You are a helpful assistant.",
)
```

```

Set some Shiny page options
ui.page_opts(
 title="Hello Anthropic Claude Chat",
 fillable=True,
 fillable_mobile=True,
)

Create and display a Shiny chat component
chat = ui.Chat(
 id="chat",
 messages=["Hello! How can I help you today?"],
)
chat.ui()

Store chat state in the url when an "assistant" response occurs
chat.enable_bookmarking(chat_client, bookmark_store="url")

Generate a response when the user submits a message
@chat.on_user_submit
async def handle_user_input(user_input: str):
 response = await chat_client.stream_async(user_input)
 await chat.append_message_stream(response)

```

The steps are:

- Initialize a `chat_client` (e.g., `ChatAnthropic()`) to interact with the LLM using `chatlas`.
- Initialize a `Chat()` instance.
- Display its UI element with `chat.ui()`.
- Decorate a `@chat.on_user_submit` function to fire when the user submits input.

While the “model=” can be anything, `Chat()` makes it especially easy to use interfaces from OpenAI, Anthropic, Google, LangChain, and Ollama.

Next, the templates coaches the user to install the required packages via the `requirements.txt` file:

```

CONSOLE TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE
phil@session-c1a7a6ae9dc1b5e09ddb-phil---positron-pro-session-wtbbj:~/py-shiny-templates$ shiny create --t
emplate chat-ai-anthropic
... Creating Chat AI using Anthropic Shiny app...
? Enter destination directory: ./chat-ai-anthropic
✓ Created Shiny app at chat-ai-anthropic

→ Next steps:
- Install required dependencies:
 cd chat-ai-anthropic
 pip install -r requirements.txt
- Open and edit the app file: chat-ai-anthropic/app.py
phil@session-c1a7a6ae9dc1b5e09ddb-phil---positron-pro-session-wtbbj:~/py-shiny-templates$

```

```
pip install -r requirements.txt
```

Above we set the `ANTHROPIC_API_KEY` environment variable in the `.env` file. If needed, you may need to restart the launching the application:

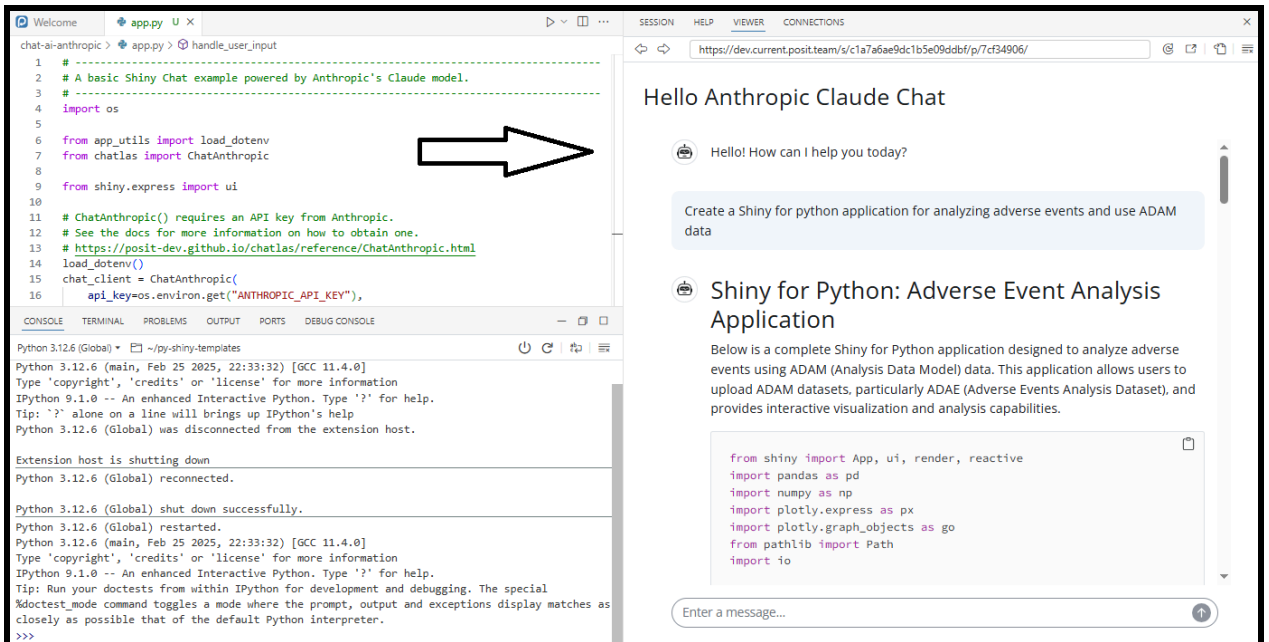




- Users then use the python-dotenv package to load them into the environment. The `python-dotenv` package will load `.env` as environment variables which can later be read by `os.getenv()`. python-dotenv is in the requirements.txt file and installed above. If needed, users can run to install python-dotenv:

```
pip install python-dotenv
```

- Lastly, run the app in Positron:



- Users can now further customize the application as needed, or deploy to the Production environment like Posit Connect. Remember that if an Application is deployed that uses ellmer to another system, ensure that the environment variable is available there as well.

## Conclusion

This paper highlights practical examples of how statistical programmers can use Shiny and GenAI for statistical programming tasks. As the use of AI increases by clinical statistical programmers, it is important organizations understand the tools that integrate with Shiny. We discussed common tools for using Shiny and GenAI as well as how to access local and provider based models. The information outlined in this paper highlights the quickly changing space of GenAI and examples of how it can be used today with Shiny by statistical programmers to add bot and interface support to LLMs.

## CONTACT INFORMATION

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

Phil Bowsher

[phil@posit.co](mailto:phil@posit.co)