PHUSE 2026 - Paper ET26

Perspective on Enterprise AI and Posit Tooling

Phil Bowsher, Posit/RStudio PBC
Sam Edwardes, Posit/RStudio PBC

Abstract

Enterprises have unique needs when it comes to AI, especially in regards to data, intellectual property, subject matter expertise, quality/SOPs, training etc. This list could be further extended with many more items. To date, I have yet to meet an enterprise that allows internal users to share such information with public LLMs such as Gemini or ChatGPT. This paper will focus on the *enterprise adoption* of AI for statistical programming and data science. We will highlight methods for managers, IT and users to surface AI tooling to aid programmers in tasks related to clinical reporting workflows. Topics related to quality, reproducibility will not be discussed but deserve much consideration.

This paper will break down the AI topics from the end-user perspective and then gradually move into IT related topics. This paper can act as a checklist for various AI topics that R/Python users and IT professionals should be aware of in statistical programming.

Introduction

Keeping up with AI seems like a full-time effort. Posit is constantly releasing new AI products and packages such as the soon to be released Posit AI. This paper will be broken into 2 parts - first it will cover end-user topics and then veer into IT topics. For info on privacy information for AI and Posit tooling, please review: https://posit.co/blog/trust-llm-tools/

LLMs make errors and domain expertise and programming knowledge is critical when working with AI coding tools as explained below. Code-first open source data science is a key pillar of Posit PBC analytical direction. Code is important for reproducibility and verification. The goal of the tools below is to *generate code* as an ability to empower statistical programmers and not the analysis of data as explained by Joe Cheng in the following video: :https://www.youtube.com/watch?v=owDd1CJ17uQ&t=878s

Further reading can be found below on these topics:

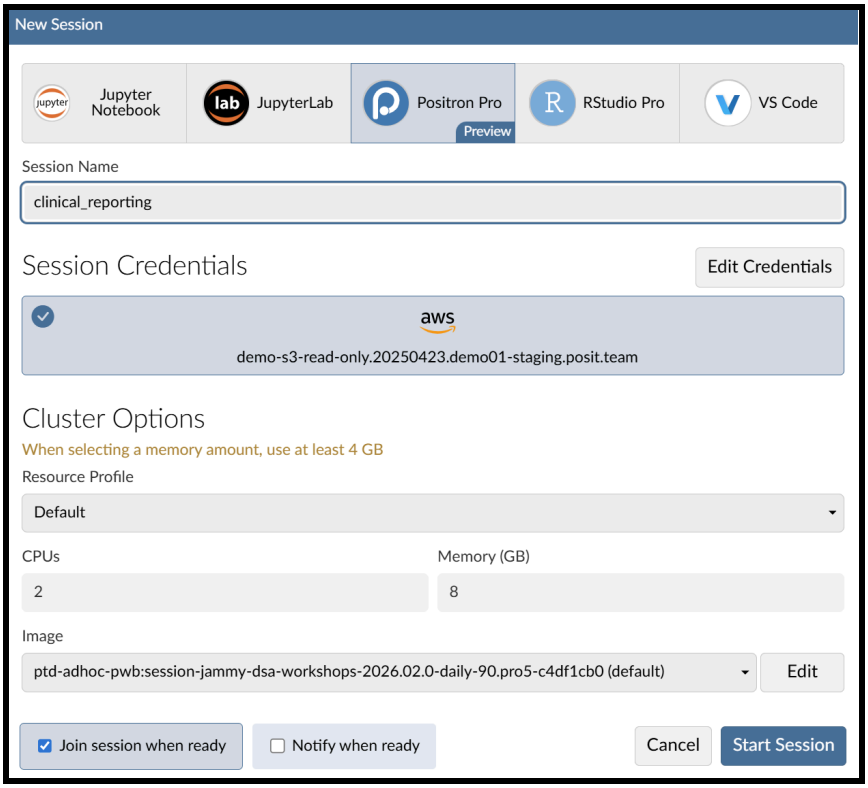
https://wesmckinney.com/blog/llms-arithmetic/

https://posit.co/blog/databot-is-not-a-flotation-device/

https://posit.co/blog/positron-ai-product-announcement-aug-2025/

This paper will introduce common tooling and then move into in-process or other soon to be released capabilities. This paper will highlight many AI features for R and Python in Positron. For an introduction to Positron, please review the following Phuse US Connect 2025 talk: https://github.com/philbowsher/Positron-Overview-for-Pharma

Positron is accessible in 2 ways: either on a local desktop for download (https://positron.posit.co/download.html) or as an Integrated Development Environment (IDE) in Posit Workbench running in a server environment:



Part 1: User Tools

Positron AI Capabilities

Positron has AI capabilities built in. There are 4 primary ways to leverage AI in Positron:

1. Positron Assistant
2. Databot
3. Code Completion
4. Open-source packages

Below we will explain the first 3 of these capabilities as they will be referenced within the paper.

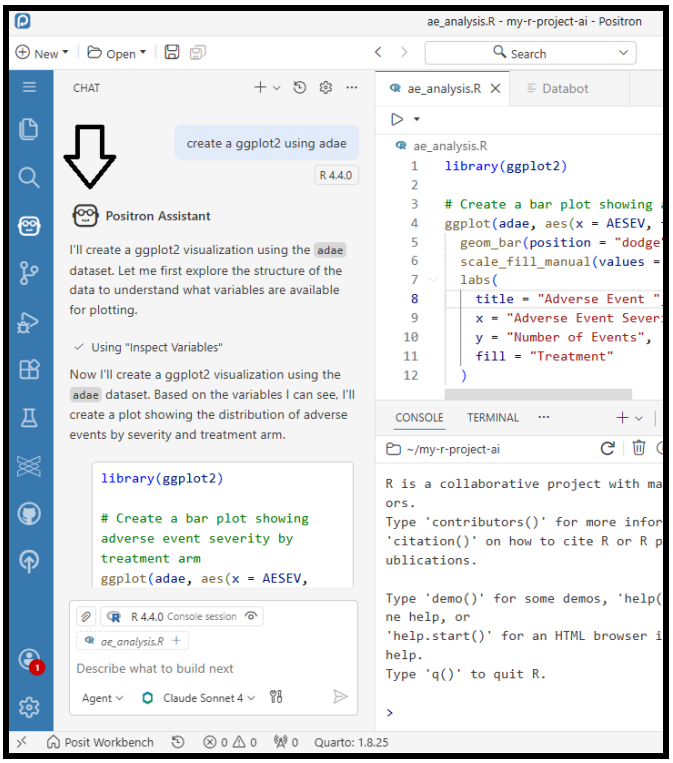
Positron Assistant

Positron Assistant is an AI coding assistant built specifically for statistical data science workflows. It is enabled by turning on the setting “positron.assistant.enable in Positron”. In Posit Workbench, IT admins can automatically enable the Positron Assistant for all users via enforced settings (https://docs.posit.co/ide/server-pro/admin/positron_sessions/user_settings.html#enforced-settings). If the setting is enabled, it may be required to restart Positron or run the *Developer: Reload Window* command in the Command Palette. To open the Command Palette, use *Ctrl+Shift+P* and then type: “Developer: Reload Window” and press Enter. More information can be found at the following links:

https://positron.posit.co/assistant.html

https://positron.posit.co/assistant-getting-started.html

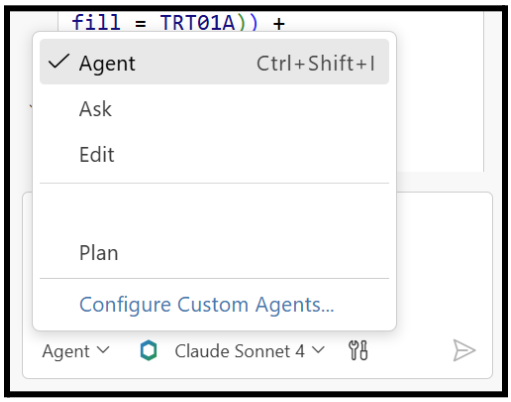
While many AI tools (Claude Code etc.) are great for software development, many are challenged with common analytical tasks. Positron Assistant is built to see and utilize in-memory data with the user’s session and therefore writes applicable code for your niche data science work and activities.



Agents

The Positron Assistant has four different agents (https://positron.posit.co/assistant-chat-agents.html) or action oriented modes as detailed below:

- **Ask**: Provides answers and code suggestions without automatic execution
- **Edit**: Suggests code changes for you to apply to files
- **Agent**: Autonomously executes work including running code and modifying files
- **Plan**: Creates detailed step-by-step plans before execution



The Positron Assistant has been curated to support data science and statistical workflows. If the task you're working on requires specific knowledge, you can create your own agent with custom instructions: https://positron.posit.co/assistant-chat-agents.html#custom-chat-agents.

Agents & Context/Skills/Specification/.md Files

For example, a user could create an agent that describes standard operating procedures for a specific clinical trial. The instructions focus on project specific knowledge/conventions/rules rather than generic advice, emphasizing the clinical trial context and the specific R ecosystem used by the user. This topic will likely become more important as more pharmas use R and Python with AI for clinical tasks where packages (pharmaverse) and other specifications are important to define per SOPs. Users could use these pharma specific *agent.md* files to preinstruct agent actions

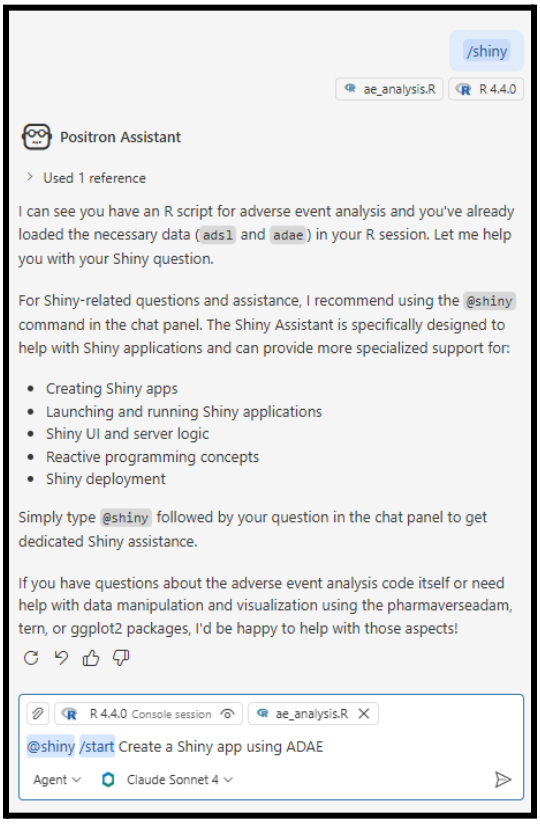
for specific pharma actions. These context files could be built using a PromptBuilder such as https://rich-iannone.github.io/talk-box/user-guide/prompt-builder.html

For organizations with Intellectual Property or proprietary instructions, *.md* files would be stored internally and shared with users. While project level instructions are possible (https://positron.posit.co/assistant-chat-instructions.html), it is on the short term roadmap to provide Positron Assistant with system level or user level instructions. This would support a scenario where an organization wants to make sure that all Positron Assistant use has a base context to apply some consistent rules or standards or alternatively, an organization might want to provide several agents that all users can choose from.

Chat participants

Out of the box, the agent understands how to perform typical data science workflows and analysis when generating code. It can be further extended with slash commands and chat participants (https://positron.posit.co/assistant-chat-commands-participants.html) that are provided by Positron extensions or manually added by users.

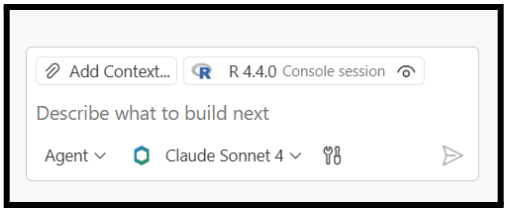
For example, users can type *@shiny* to invoke the Shiny participant provided by the Shiny extension (https://open-vsx.org/extension/posit/shiny) which is included by default (https://positron.posit.co/extensions.html). Extensions can also provide custom slash commands like */start* provided by the Shiny extension.



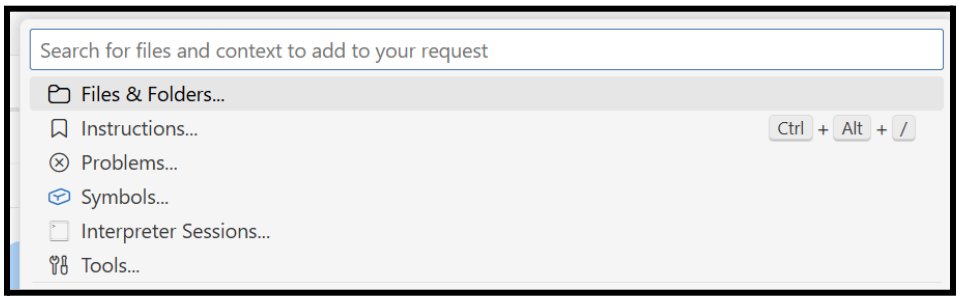
Context

One feature that sets the Positron Assistant apart from other coding agents is its ability to inspect your coding environment. The Positron Assistant is able to see your in-memory data (e.g. data frames) plots in the plotting pane, console history, and information about your workspace (e.g. R version, file structure, etc.).

Users can attach additional context to the chat through the **paperclip** icon or by typing *#<file|folder|symbol>*. Below details the steps in the Positron Assistant:



The Context section will open the following dialog:



See https://positron.posit.co/assistant-chat-context.html for more details on all of the context Positron Assistant has access to.

Custom instructions

Users can create project-wide custom instructions to provide information about the project. These can be checked into version control so all collaborators get a similar starting point for context. For example, you can create a *positron.md* file in the root of your project.

```
None
# positron.md

This project using R 4.4.0 to statistically analyze the results of the clinical
trial. BioCondcutor is the primary repositroy for R packages. We also use an
internally developed package `ourRPacakge` that can be obtained from Github

<!-- additional instructions below... -->
```

See https://positron.posit.co/assistant-chat-instructions.html for more information.

Databot

Databot is an AI assistant designed to dramatically accelerate exploratory data analysis for data scientists and programmers fluent in Python or R. To install Databot, in Positron, choose the Extensions view from the Activity Bar on the left or use the keyboard shortcut Ctrl-Shift-X and search for “Databot”. Install the Databot extension. Then in Positron, open the Command Palette (Ctrl-Shift-P) and run “Open Databot in Editor Panel”.

https://positron.posit.co/databot.html

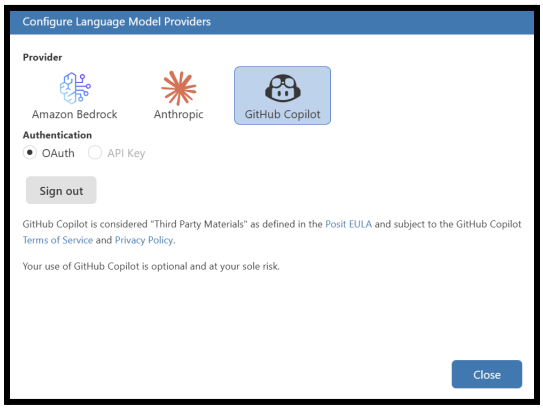
Databot will provide you with several suggested prompts to start. Here is an example prompt you could use with Databot to help understand a new data set.

"Use R to load all of the ADaM data in the /data folder. Summarize each dataset (adsl, adae etc.) and identify if there's any relationships between adverse events and severity."

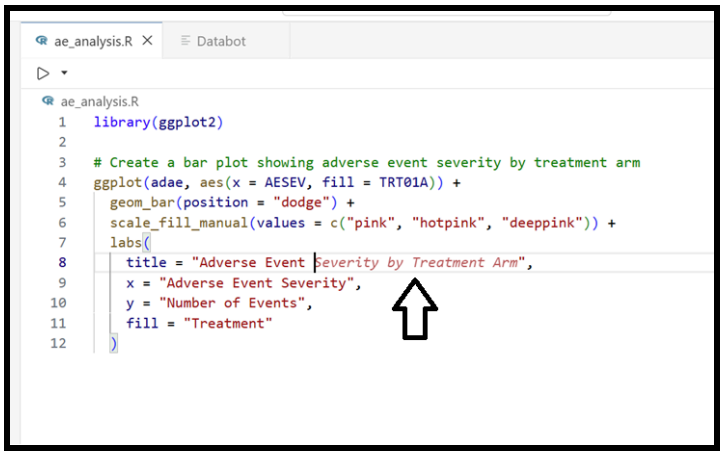
Code Completion

One of the first GenAI tools available was GitHub Copilot, which provides AI models and code completions for programming. Please note that GitHub Copilot can be used for agentic coding as well as will be seen late rin the paper when discussing Model Providers. Below we will highlight code completion.

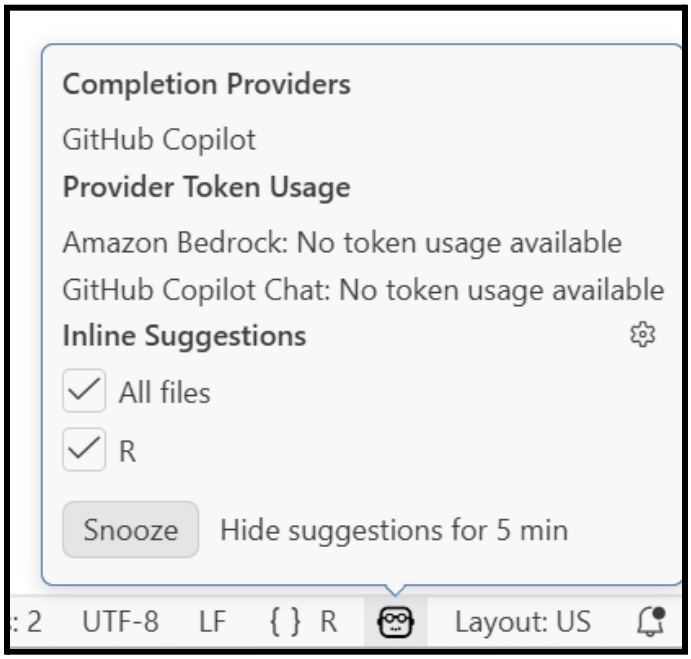
Below we will highlight the steps for Positron to enable code completion via *Github Copilot*. Go to the Positron Assistant and click on "Configure Model Providers…":



Users will then need to authenticate into Github and enable the connection. Now users will have code completions in their R and Python scripts such as:



Users can check to see if Github Copilot completion provider is running by clicking the Positron AI icon:



GitHub Copilot is also available in RStudio: https://docs.posit.co/ide/user/ide/guide/tools/copilot.html. Please note it is possible to authenticate into multiple model providers while using Positron.

AI Information

Many pharmaceutical companies are using Agents, .md files and Claude Code for various drug development tasks as seen here for Pharmacokinetic/Pharmacodynamic (PKPD) use-cases:
https://www.aripritchardbell.com/blog/2025-10-16-ACoP

Below we will review various important topics for AI such as deployments, tool calling, MCP (Model Context Protocol) and open-source tools.

MCP (Model Context Protocol)

MCP use-cases

Organizations utilize MCP to connect LLMs to other tools. Similar to how people use ODBC to connect R and Python to databases, MCP provides a standardized interface to connect LLMs (like Claude Code or Positron Assistant) to external tools, context, and data. You can read more about MCP here: https://modelcontextprotocol.io/. Here are several examples of how MCP is used:

- Since R and Python utilize in-memory data, an MCP server can be used to access the session state in real time. See https://posit-dev.github.io/mcptools/reference/server.html for more information. The Positron Assistant has access to your session's context out of the box, but if you want this context available to other large language model tools, MCP can be used.
- Models are trained at a certain point in time and may or not have access to the most recent documentation for popular R and Python libraries. The Context 7(https://github.com/upstash/context7) MCP server can be used to add the latest docs to a model's context.
- The GitHub (https://github.com/github/github-mcp-server) MCP server can be used to take actions (e.g. close an issue) or add context to your LLM session (e.g.add details from an issue).

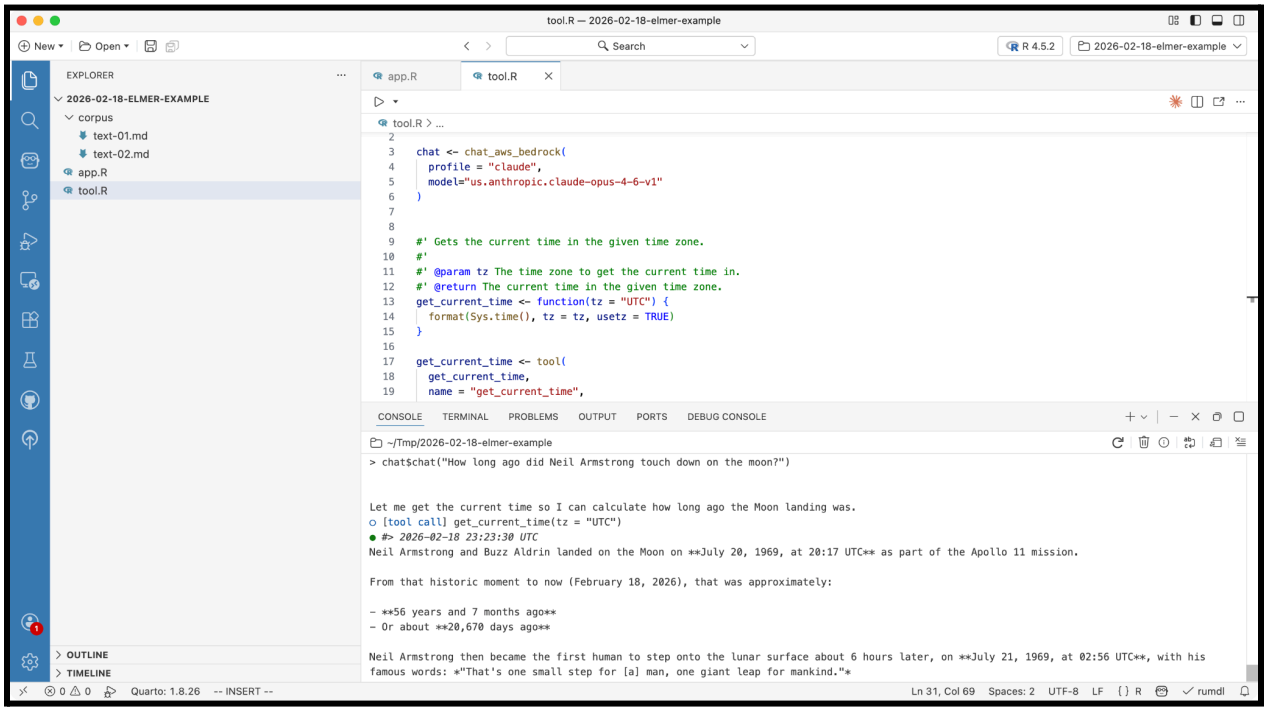
Tool calling

Sometimes an MCP server might be more than needed. Users may have a custom function for a large language model to be able to use, but do not want to create an entire MCP server for the use case. Both ellmer (https://ellmer.tidyverse.org/articles/tool-calling.html) and chatlas (https://posit-dev.github.io/chatlas/) have built-in support for defining custom tools.

An example for ellmer:

```
None
library(ellmer)

chat <- chat_aws_bedrock(
  profile = "claude",
  model="us.anthropic.claude-opus-4-6-v1"
)


#' Gets the current time in the given time zone.
#'
#' @param tz The time zone to get the current time in.
#' @return The current time in the given time zone.
get_current_time <- function(tz = "UTC") {
  format(Sys.time(), tz = tz, usetz = TRUE)
}

get_current_time <- tool(
  get_current_time,
  name = "get_current_time",
  description = "Returns the current time.",
  arguments = list(
    tz = type_string(
      "Time zone to display the current time in. Defaults to `\"UTC\"`.",
      required = FALSE
    )
  )
)
```

```
chat$register_tool(get_current_time)

chat$chat("How long ago did Neil Armstrong touch down on the moon?")
```

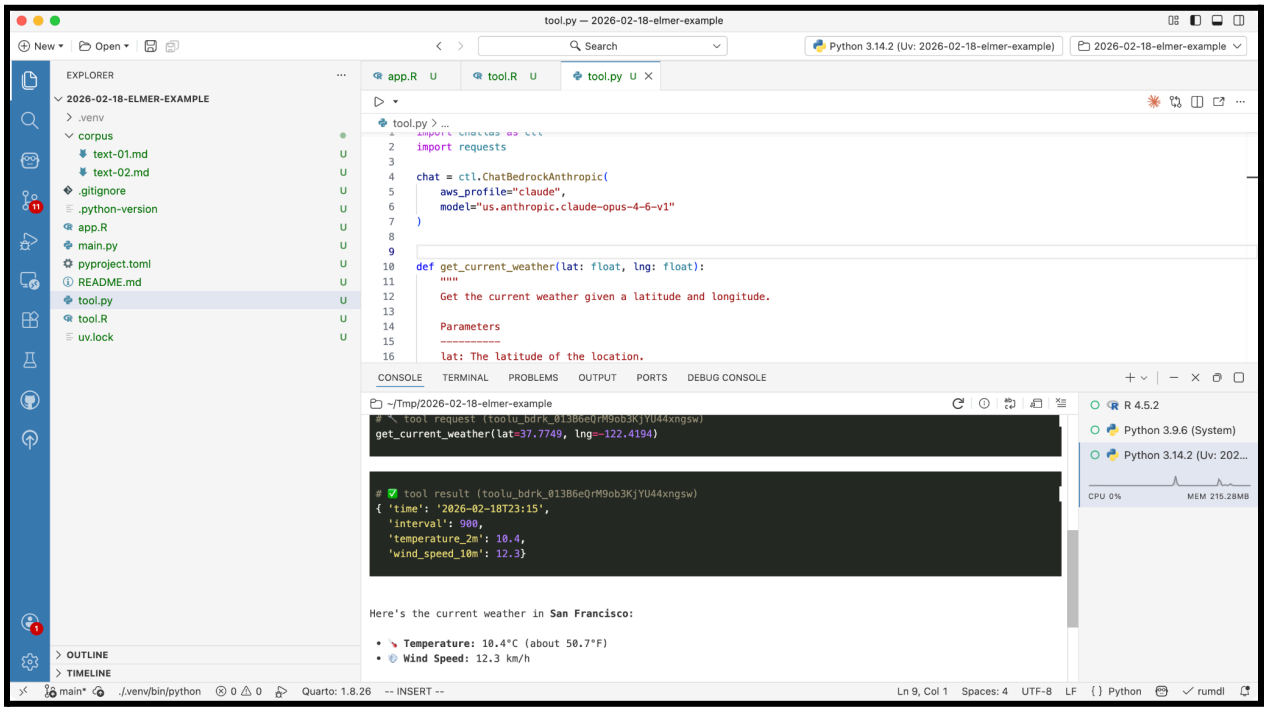


An example for chatlas:

Python

```python
import chatlas as ctl
import requests

chat = ctl.ChatBedrockAnthropic(
    aws_profile="claude",
    model="us.anthropic.claude-opus-4-6-v1"
)


def get_current_weather(lat: float, lng: float):
    """
    Get the current weather given a latitude and longitude.

    Parameters
    ----------
    lat: The latitude of the location.
    lng: The longitude of the location.
```

```
    """
    lat_lng = f"latitude={lat}&longitude={lng}"
    url =
f"https://api.open-meteo.com/v1/forecast?{lat_lng}&current=temperature_2m,wind_
speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"
    response = requests.get(url)
    return response.json()["current"]

chat.register_tool(get_current_weather)

chat.chat("How's the weather in San Francisco?")
```



Open-Source Tools

In addition to the Positron AI tools above, users can extend AI capabilities by utilizing open-source packages for AI.

The primary R package is ellmer: https://github.com/tidyverse/ellmer

The primary Python package is chatlas: https://posit-dev.github.io/chatlas/

Users will need to obtain an API key from the model provider (e.g. OpenAI, Anthropic, AWS Bedrock) and then store that securely in the environment. Users should set an environment variable rather than using the API key directly in users code. In R, users can use the R package usethis (https://usethis.r-lib.org/), to set up the API keys in Positron. Users can open the *.Renviron* for editing with usethis::edit_r_environ(). Add the API key(s) to the *.Renviron*, for example:

```
Shell
OPENAI_API_KEY=my-api-key-openai-xyz
ANTHROPIC_API_KEY=api-key-anthropic-xyz
GOOGLE_API_KEY=api-key-google-xyz
```

When using Python and chatlas, users can store the environment variables in a *.env* file. To create a new *.env* file in Positron, open the Command Palette and type “New File” and enter *.env*. In the file, add the following:

```
Shell
OPENAI_API_KEY=my-api-key-openai-xyz
ANTHROPIC_API_KEY=api-key-anthropic-xyz
GOOGLE_API_KEY=api-key-google-xyz
```

Users then use the *python-dotenv* (https://pypi.org/project/python-dotenv/) package to load them into the environment. To install python-dotenv execute:

```
Shell
pip install python-dotenv
```

Users then add the following to the top of the script or application:

```
Python
from dotenv import load_dotenv

load_dotenv()

# The rest of your code goes here.
```

Below is an example of using Anthropic models to create a Shiny application that can answer questions about a knowledge corpus added to the models context.

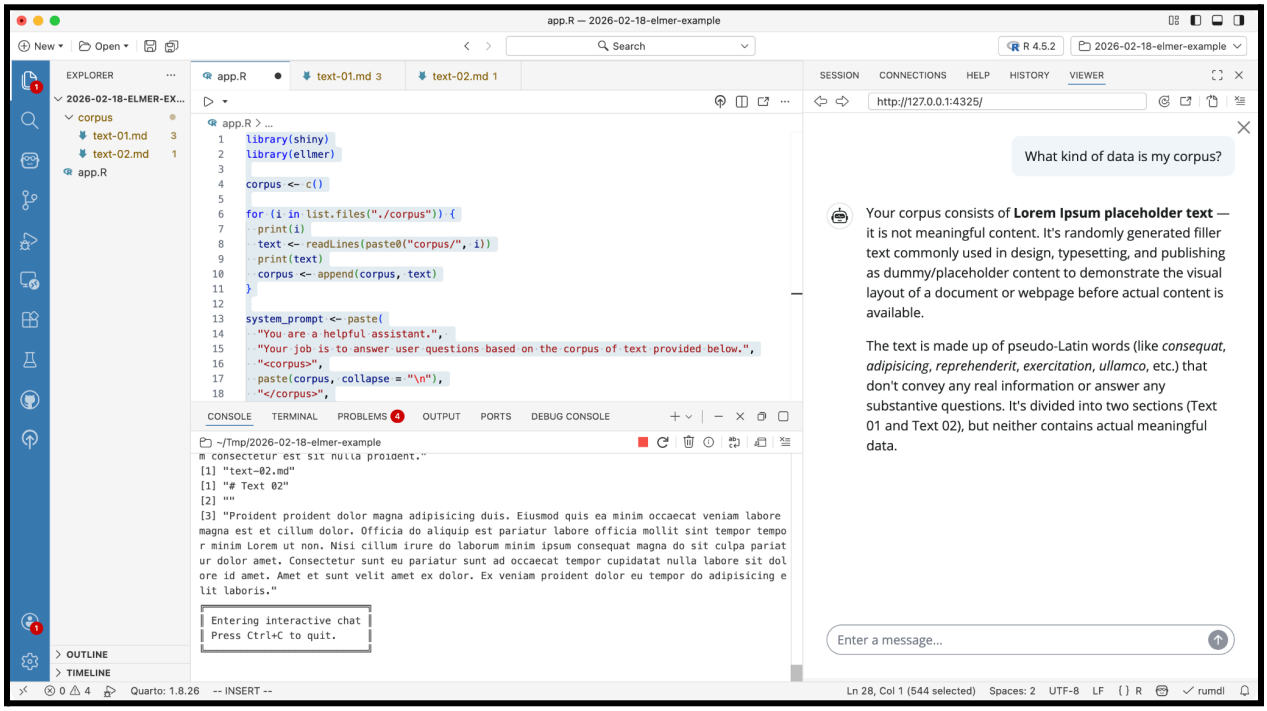
After connecting to the API key, users are ready to use *ellmer* connected to Bedrock (explained below) to create a Shiny app. Below we will use Bedrock with an Anthropic Claude model.

After securely connecting to Bedrock, each connection will start by creating a new chat object like as below in R:

```
None
library(ellmer)

corpus <- c()

for (i in list.files("./corpus")) {
```

```
  text <- readLines(paste0("corpus/", i))
  corpus <- append(corpus, text)
}

system_prompt <- paste(
  "You are a helpful assistant.",
  "Your job is to answer user questions based on the corpus of text provided
below.",
  "<corpus>",
  paste(corpus, collapse = "\n"),
  "</corpus>",
  collapse = "\n"
)

chat_openai(system_prompt, model = "gpt-4o-mini")

live_browser(chat)
```

This code will be executed in Positron as below:



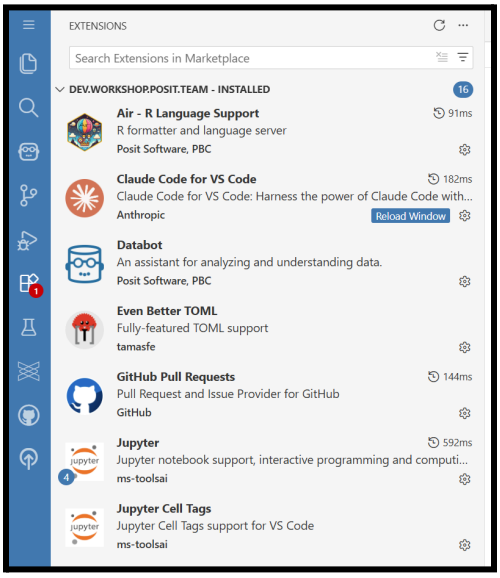
There are many great open-source packages for AI with R and Python. For a review of such tooling for Shiny, please see: https://github.com/philbowsher/Shiny-LLMs-Landscape-and-Applications-in-Pharma. The paper covers https://posit-dev.github.io/querychat/ which facilitates natural language exploration of tabular data, powered by SQL and large language models (LLMs). Moreover, https://posit-dev.github.io/shinychat/ is discussed as a way to bring LLM-powered chat interfaces to Shiny apps. For voice driven analysis, please see https://github.com/tidyverse/ggbot2

Claude Code & Positron Integration

In addition to *the Positron Assistant, Databot*, and *code completion,* Positron is compatible with AI extensions on OpenVSX such as Google Gemini, Claude Code, AWS Q, and Continue.dev. Below outlines the process for installing *Claude Code* within a Positron environment, utilizing Amazon Bedrock as the backend provider, and utilizing Posit Workbench for Positron access.

Install the Claude Code Extension

Install the Claude Code OpenVSX Extension: https://open-vsx.org/extension/Anthropic/claude-code



Install the Claude Code CLI:

```shell
curl -fsSL https://claude.ai/install.sh | bash
```

Install Node JS:

```shell
curl -fsSL
https://raw.githubusercontent.com/mklement0/n-install/stable/bin/n-install |
bash -s 24
```

Users will need to quit the current Positron session in Workbench, and then launch a new session. This is required for Positron to load the new environment variables that were set in ~/.bashrc as part of installing Node JS.

Typically, these steps are performed by an administrator so that each individual user does not have to install Claude Code. In the future, the process of using the Claude Code extension in Positron should be simpler. There are known issues with the Claude Code VS Code extension at this time:

- anthropics/claude-code#11868 (https://github.com/anthropics/claude-code/issues/11868)
- anthropics/claude-code#10506 (https://github.com/anthropics/claude-code/issues/10506)
- anthropics/claude-code#8757 (https://github.com/anthropics/claude-code/issues/8757)
- anthropics/claude-code#8410 (https://github.com/anthropics/claude-code/issues/8410)

Configure Claude Code to use Bedrock

This example we use in Amazon Bedrock as a model provider. Claude Code supports several model providers, you can choose the one that you prefer or have access to.

Disable the Claude Code login prompt in your Positron user settings:

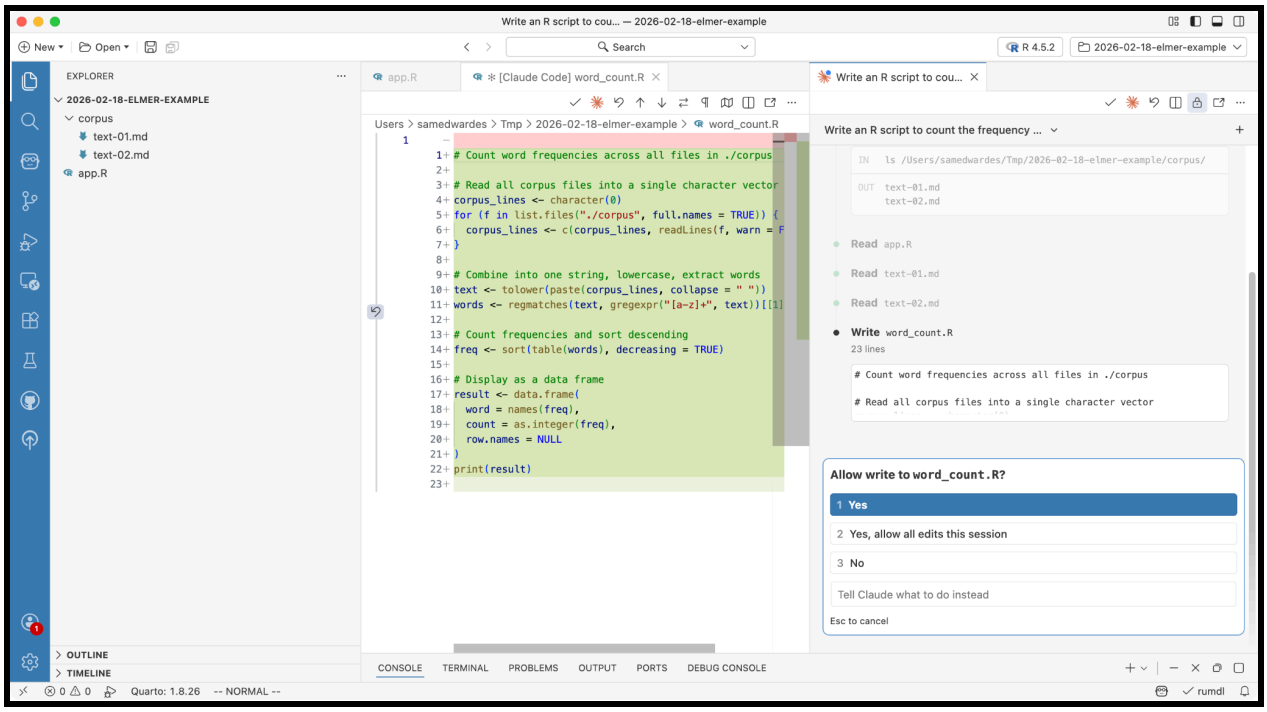
```JSON
{
  "claudeCode.disableLoginPrompt": true
}
```

Configure Claude Code to use AWS Bedrock. This example will use single-signon. See https://code.claude.com/docs/en/vs-code#use-third-party-providers for more details on configuring AWS Bedrock. Add the following to ~/.claude/settings.json:

```JSON
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "env": {
    "CLAUDE_CODE_USE_BEDROCK": "1"
  }
}
```

Use Claude Code

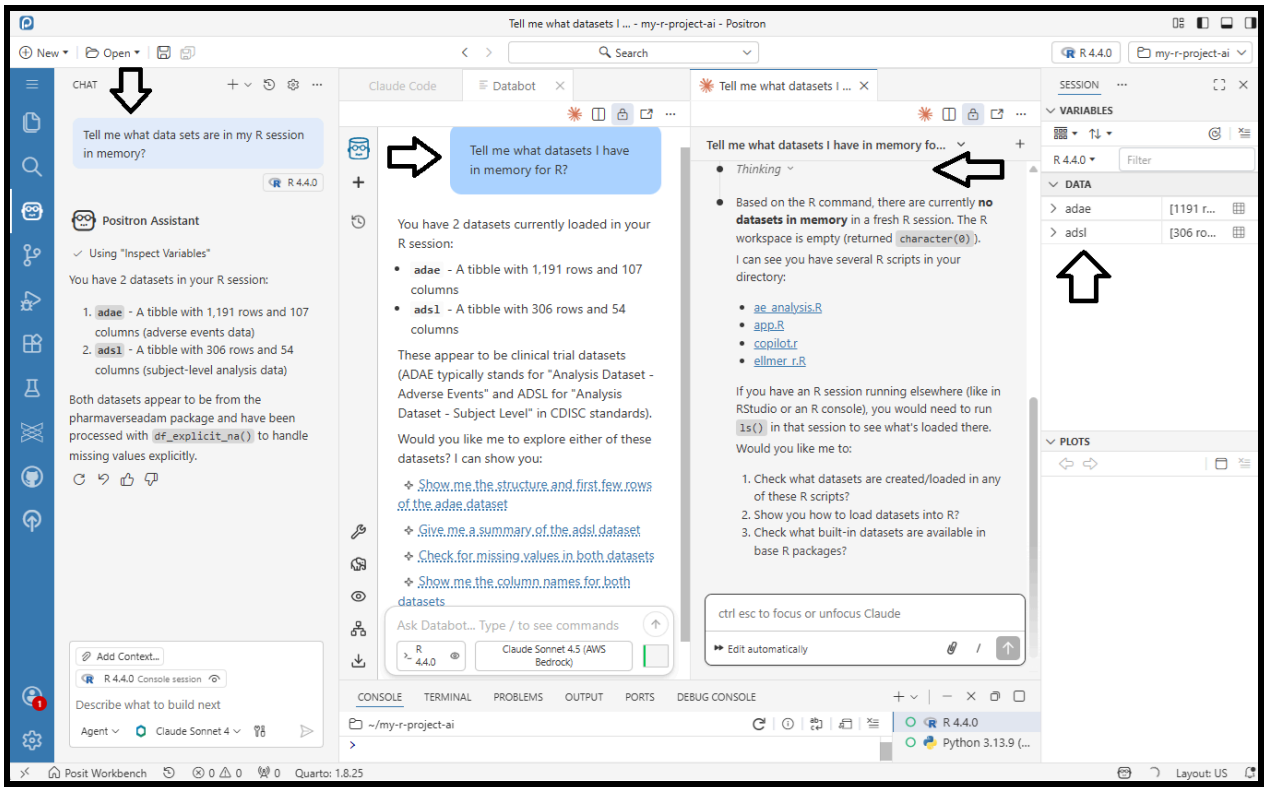
Open the Command Palette, and type “Claude Code: Open in New Tab”. You can now use Claude code in Positron.



Using MCP to add your R session to Claude's Context

One challenge to keep in mind for Claude Code out of the box in Positron: it will not be able to see the variables in your current session (e.g. a dataframe you have loaded). Hadley Wickham mentioned that "R is a fantastic environment for the rapid exploration of in-memory data" but Claude Code will prefer the data be on disk.

Below is a side by side comparison on Positron Assistant, Databot and Claude Code. All 3 tools were asked exactly the same question, **"Tell me what datasets I have in memory for R?"** and the Assistant and Databot returned the correct answer but Claude Code got it wrong! Claude Code can not find the R datasets because of a process boundary issue. Claude Code can not access the active R session's RAM even though we are using it in Positron's terminal or Extension. Claude code can see our flat files and saved .R files, but it can not see the active and wrangled dataset in our R console! Below is an example:



To help Claude Code, users will need to venture back into our previous topic, MCP. If wanted, users could use the *btw* (https://github.com/posit-dev/btw) and *mcptools* (https://github.com/posit-dev/mcptools) to add tools to Claude Code to connect it to the R and or Python session. For more information on these tools and Claude Code, please see:
https://posit-dev.github.io/btw/reference/mcp.html
https://cran.r-project.org/web/packages/mcptools/vignettes/server.html

Part 2: Admin Topics

Enforcing Settings with Positron

Positron is a very flexible tool. It can integrate with many different model providers. In enterprise settings, administrators may want to restrict the model providers that users can use to a subset of approved tools. This is possible when running Positron in Posit Workbench through enforced settings (https://docs.posit.co/ide/server-pro/admin/positron_sessions/user_settings.html#enforced-settings). In the example below, an administrator disables the Positron Assistant so that users cannot leverage large language models.

Add the following to /etc/rstudio/enforced-settings.json:

```JSON
{
  "positron.assistant.enable": false
}
```

Add the following to /etc/rstudio/profiles:

```
Shell
[*]
positron-enforced-settings = /etc/rstudio/enforced-settings.json
```

MCP (Model Context Protocol) & Posit Enterprise Tooling

MCP in Positron

While more of an user feature, Positron Assistant can use MCP servers and the integration is mostly inherited from VS Code. More information is here: https://code.visualstudio.com/docs/copilot/customization/mcp-servers

Positron does not currently support the MCP marketplace, so users have to install manually in an *mcp.json* file as explained in the link above. Users can search and explore MCP servers in Positron via the Command Palette (Ctrl+Shift+P) and typing *MCP: Add server*.

Posit Package Manager MCP Server

Posit Package Manager's Model Context Protocol (MCP) server is an authoritative backend service, exposing package information, capabilities, and metadata to AI tools and coding assistants. This helps users and AI agents to know exactly what packages and dependencies are allowed and how they work within the pharmaceutical organizations established policies. For example, if an organization restricts or blocks the use of a certain (non-tested or qualified) package, AI agents that use the MCP server can avoid recommending that package to users. Users can add the Package Manager MCP server to user settings by opening the Command Palette (Ctrl+Shift+P) and runing "MCP: Add Server". Read more about this capability here: https://docs.posit.co/rspm/admin/ai/mcp.html

Posit Connect for Deploying & Hosting MCP Servers

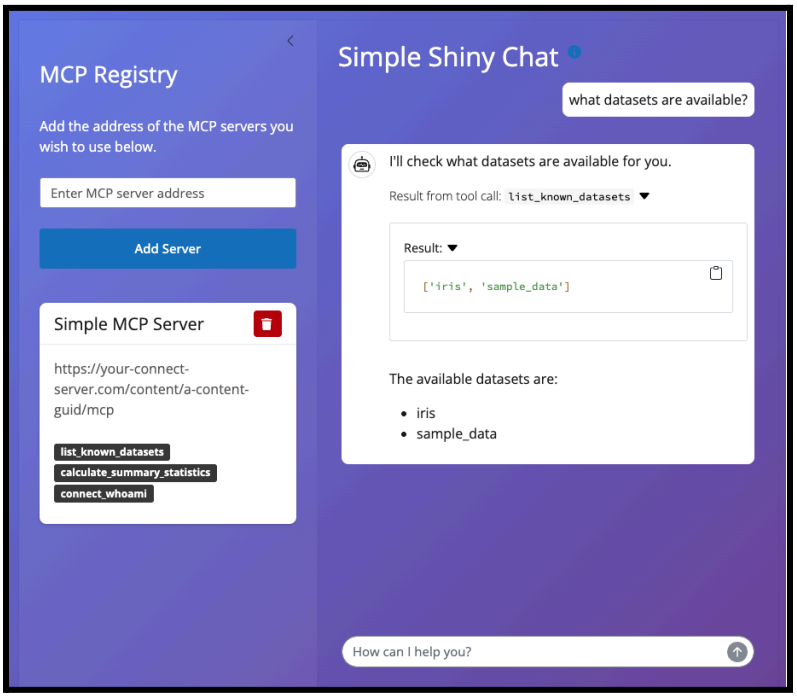
Users and organizations can deploy MCP servers to Posit Connect to enable AI clients across the enterprise to access shared and approved tools and resources. Below are some connected examples, showing how MCP can enable interactive AI-powered applications on Posit Connect. Below are examples of MCP servers on Connect. The code for these examples are here: https://github.com/posit-dev/connect-extensions

1. https://github.com/posit-dev/connect-extensions/tree/main/extensions/simple-mcp-server

The extension below highlights Posit Connect's ability to host MCP servers that can be consumed by AI tools or assistants as well as other MCP clients, enabling LLMs to access and execute tools remotely.

2. https://github.com/posit-dev/connect-extensions/tree/main/extensions/simple-shiny-chat-with-mcp

This Simple Shiny Chat extension is an example designed to be paired with MCP servers deployed on Posit Connect, supporting an ecosystem where AI tools and assistants can dynamically access and run other capabilities.



Learn more about Posit Connect and MCP Servers here:

https://docs.posit.co/connect/user/mcp-servers/

Model Providers/Platforms

Positron's AI capabilities are meant to work with many different model providers and platforms. You can see the current list here: https://positron.posit.co/assistant-provider-info.html. Below we highlight several model providers that are commonly used in enterprise settings. Please note it is possible to login into various model providers at the same time for a mix of different capabilities.

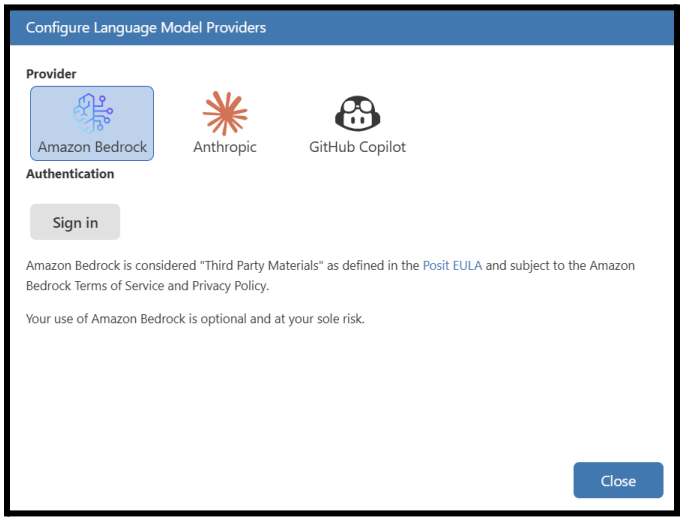
AWS Bedrock

Most organizations utilize Foundation Model Platforms, such as AWS Bedrock, Azure AI Foundry, and Google Vertex AI. These platforms help with Enterprise topics such as keeping data private, allows users to pick different models, and are pay as you go (tokens). Below we will discuss AWS Bedrock with Posit tooling. To learn more about AWS Bedrock: https://aws.amazon.com/bedrock/

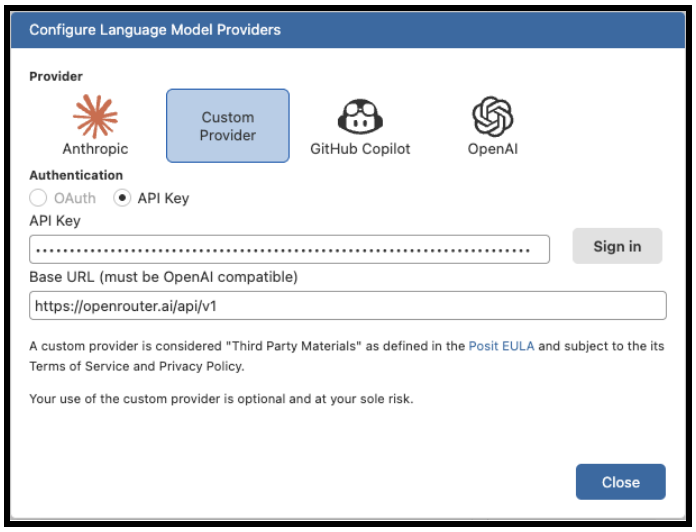
Positron Pro 2025.10.1 introduced the ability to connect to more AI models, such as Bedrock, to enable enterprise users to keep their chats within their AWS environment. To enable Positron Assistant with AWS Bedrock, users add the following specific user-level Positron settings in settings.json:

```JSON
{
    "positron.notebook.enabled": true,
    "positron.assistant.enabledProviders": ["amazon-bedrock"],
}
```

Then, users can open the Command Palette, run the command “Positron Assistant: Configure Language Model Providers” to open the Provider Configuration dialog. Users then Select “Amazon Bedrock” from the list of providers and “Sign in”. For most Pharma users, in Posit Workbench, IT administrators will configure Workbench Managed Credentials for AWS, making the login almost instant. Users can also login via the Positron Assistant interface.



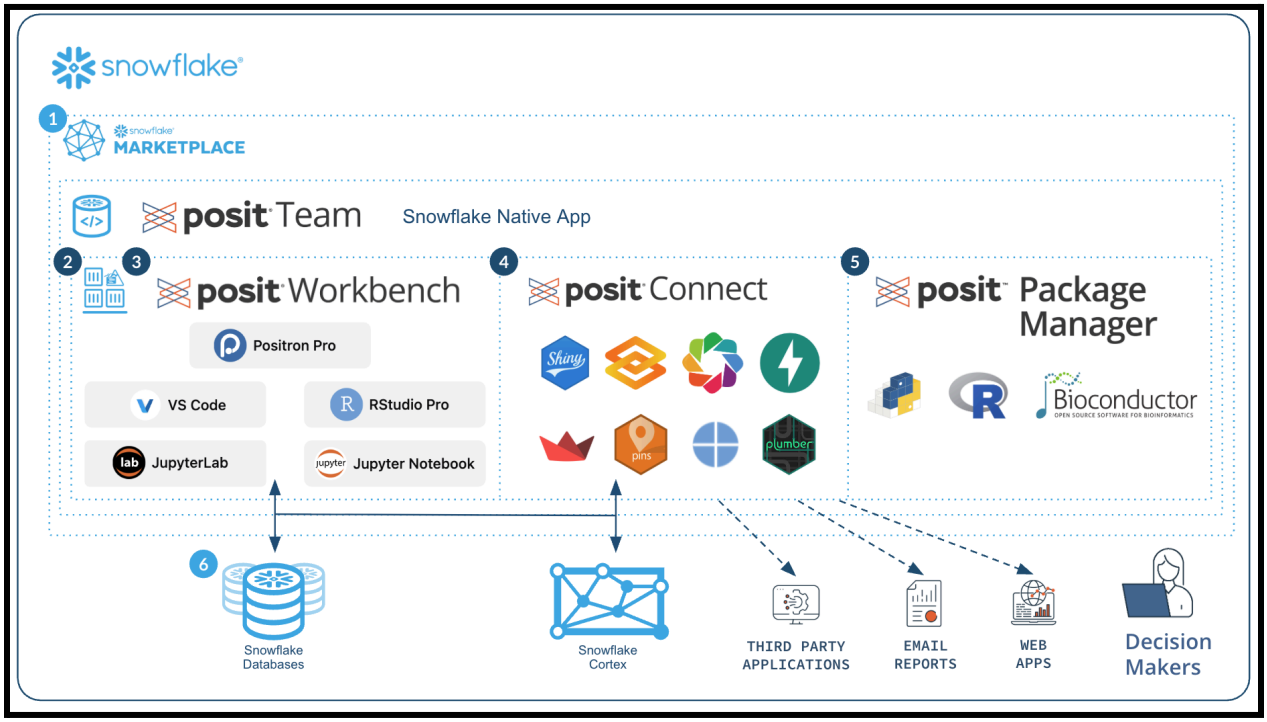
Beyond Bedrock, there is also experimental support for Custom Providers:

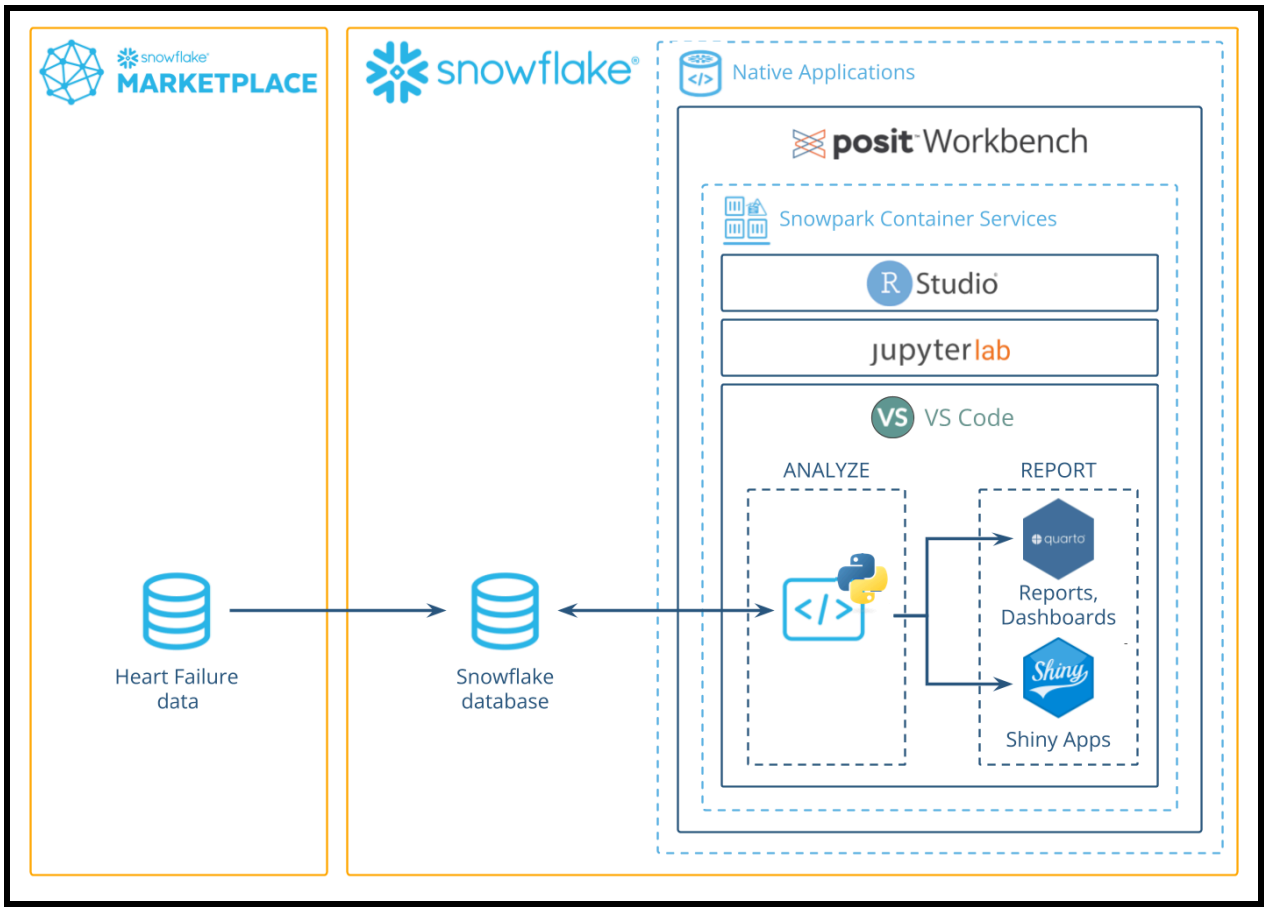


For pharma, Custom Providers may be interesting as all code and prompts remain within the private or even offline infrastructure, (HIPAA, GDPR etc. requirements) without transmitting data to external services or countries.

Snowlfake Cortex

Many enterprises use Databricks and or Snowflake for managing data, often real-world and or genomics. There are many ways to connect Posit Tooling with these platforms as the example with Snowflake below:





https://www.snowflake.com/en/developers/guides/analyze-data-with-python-using-posit-team/

https://www.snowflake.com/en/developers/guides/analyze-data-with-python-using-posit-workbench-and-snowflake/

https://posit.co/blog/posit-snowflake-product-announcement-aug-2025/

These platforms also offer many enterprise AI capabilities such as Databricks Mosaic and Snowflake Cortex. Below we will highlight how to connect Positron to Snowlfake Cortex.

Snowflake Cortex gives you instant access to industry-leading LLMs trained by researchers at companies like Anthropic, Mistral, Reka, Meta, and Google. With Snowflake Cortex, the organization's user data stays within Snowflake, providing performance, scalability, and governance.

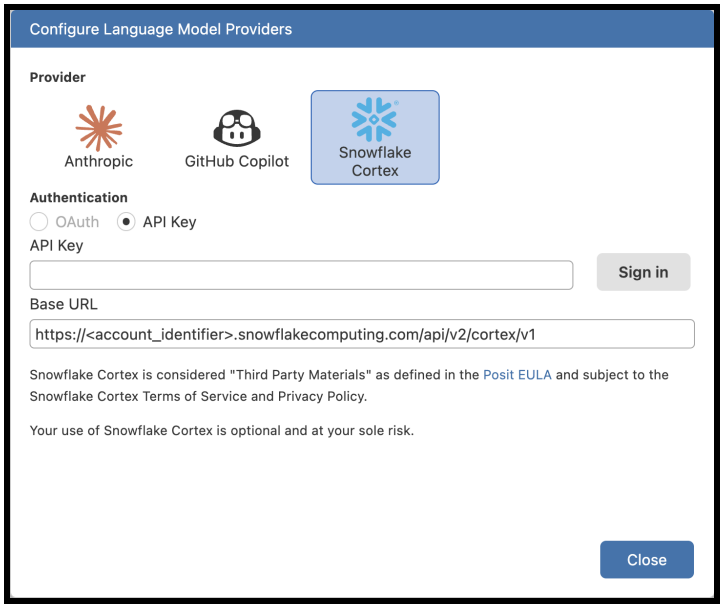
https://posit.co/blog/ai-assisted-ai-posit-snowflake-cortex/

Users can connect Snowflake Cortex to Positron Assistant. Positron Assistant requires a Snowflake account with Cortex access, as well as a Snowflake account identifier and programmatic access token (PAT). Run the command Preferences: Open User Settings (JSON) and add the following to your settings file:

```JSON
{
    "positron.notebook.enabled": true,
    "positron.assistant.enabledProviders": ["snowflake-cortex"]
}
```

To add Snowflake Cortex as a language model provider, users add the SNOWFLAKE_ACCOUNT ID to the positron.assistant.providerVariables.snowflake setting by running the command “Positron Assistant: Configure Language Model Providers” via the Command Pallet.

Now users can select Snowflake Cortex as the model provider in the Model Providers interface. Lastly, users paste the Snowflake Cortex PAT into the API key field, and click Sign in:



You can read more about these steps here: https://positron.posit.co/assistant-getting-started.html

Conclusion & Contact

The information above highlights the evolving AI ecosystem of Enterprise statistical computing and data science. This paper highlighted applicable AI use-cases for Posit Enterprise tooling such as Posit Workbench and Positron, Posit Connect and Posit Package Manager. As AI’s use in pharma increases, Enterprise capabilities for highly regulated use-cases will likely incorporate many strategies for open-source languages like R and Python. This paper covered many topics that span open-source to Enterprise features to help modernize clinical processes with innovative new features and capabilities.

Phil Bowsher

phil@posit.co

https://github.com/philbowsher