

PHUSE 2025 - Paper ET20

Positron: Overview for Pharma

Phil Bowsher, Posit/RStudio PBC

ABSTRACT

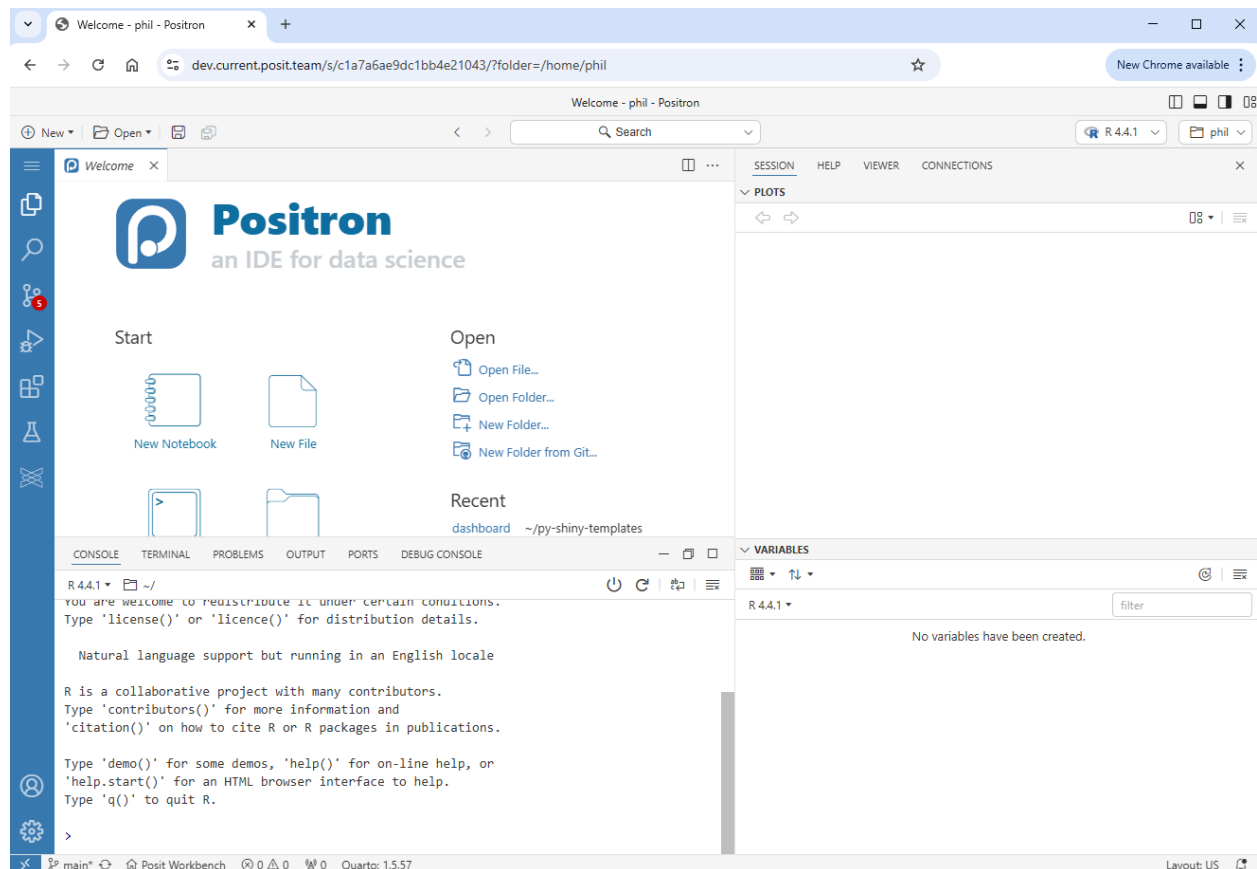
This paper will introduce Positron, a next generation data science integrated development environment (IDE) currently in Public Beta. The main Positron site for reference is: <https://positron.posit.co/>

The Positron IDE is an extensible, polyglot tool for statistical programming and analyzing data. Positron is built on [Code OSS](#) and incorporates many of the data-science centric features of RStudio while also including text editor features and extensibility from Visual Studio Code (VS Code). The goal of this paper is to introduce statistical programmers to Positron for clinical reporting using open-source. You can find all of the examples in this paper as well as the Phuse Conference presentation slides here: <https://github.com/philbowsher/Positron-Overview-for-Pharma>

Positron is currently in Public Beta, and is available for free on desktop and as a Preview feature within Posit Workbench. You can download Positron Desktop here: <https://positron.posit.co/download>

Posit Workbench installation is here: https://docs.posit.co/ide/server-pro/getting_started/installation/installation.html

Positron doesn't bundle either Python or R so it is important to check at least one of them is installed ahead of time. Since most pharmaceutical organizations manage Statistical Programmers via a server based environment, the focus of this paper will be on Positron via Posit Workbench. Below is an image of Positron accessed in Posit Workbench via a browser:



INTRODUCTION

The RStudio IDE was released in 2011, making it easier to do statistical analysis in R. While support for languages

beyond R exists in RStudio, such as running Python scripts or using Reticulate, many non-R users use VS Code for software development. However, VS Code is an extensible code editor with a focus on software development, not data science or statistical programming. For example, VS Code doesn't have a dedicated R console or first-party support for R. VS Code extensions are added by users to incorporate new features and create the IDE environment as needed. Extensions are by design, limited in the capabilities they can provide as well as the UI they can add or change. We are seeking to not only create a "batteries included" experience for doing data science in Positron, but also include novel UI and capabilities to make data scientists more productive.

Positron was made to focus on statistical reporting and data-oriented work, such as including key items like a dedicated console, data explorer, plots pane, and output viewer. Rather than adding these needed statistical "extensions," Positron provides them out of the box. These features are part of Positron core so that they can be shared among all languages. Language support, like R and Python, is also implemented natively in Positron while still making use of user-supplied language runtimes. Below, we will introduce Positron and provide some examples, such as creating a Shiny for a Python application.

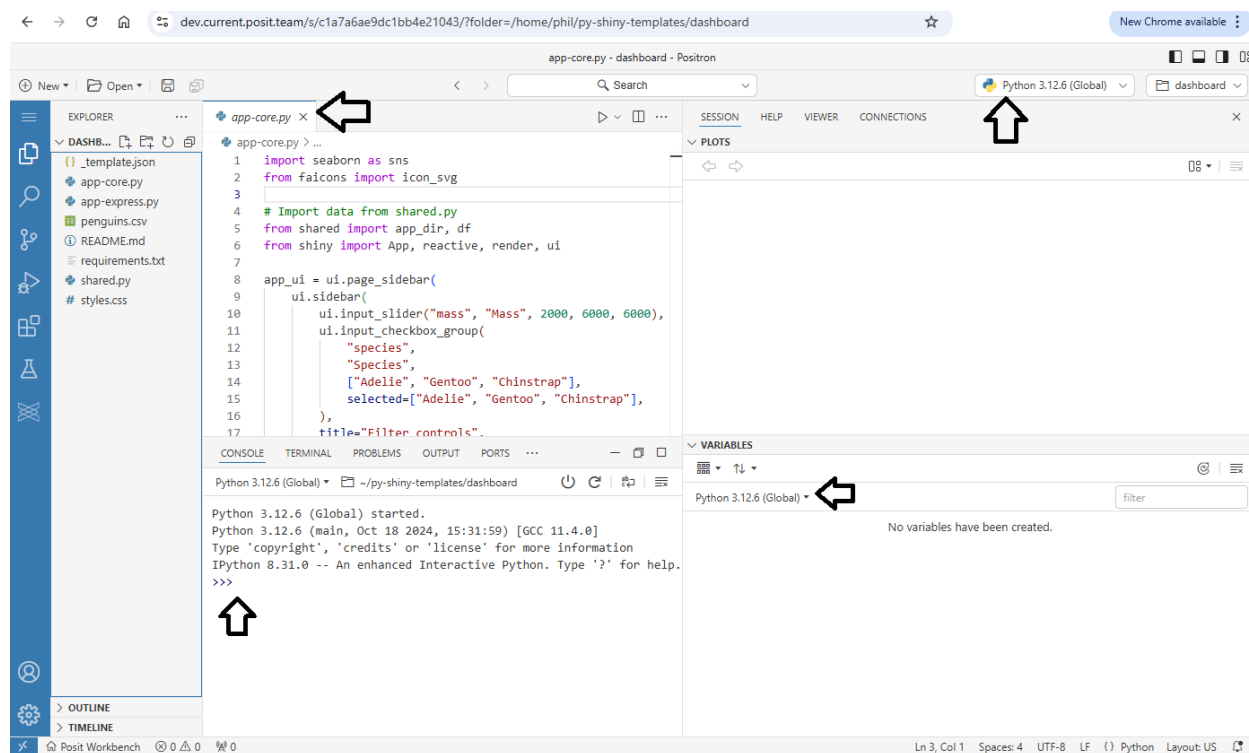
POLYGLOT IDE & MULTI-LANGUAGE SUPPORT

Many pharmaceutical organizations support a data science toolbox of various languages used for several use-cases. These environments often include languages like R, Python, Julia, Stan, SQL, Javascript etc. Positron has first-class, built-in support for R and/or Python with extensibility options for other languages planned in the future. Positron adds the core data analysis components while still providing extensions to provide further customizations as needed.

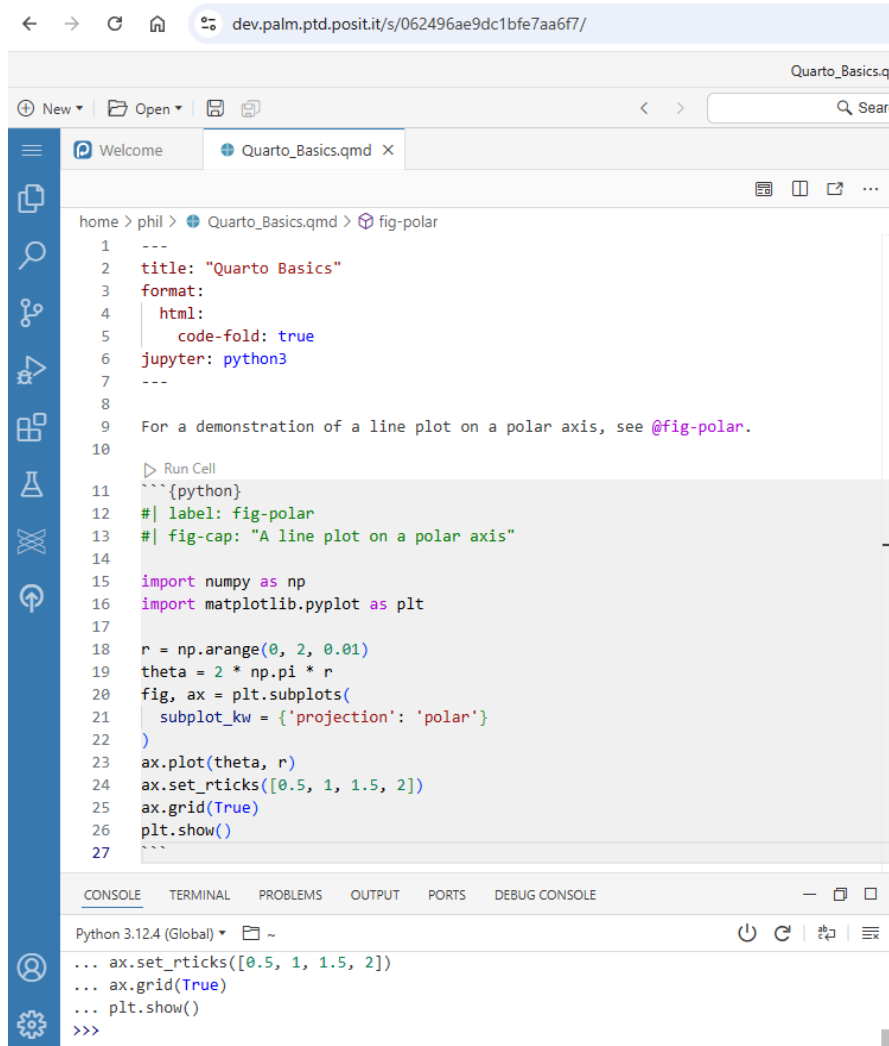
Some clinical reporting tasks are becoming Python based and many of these use-cases are highlighted in the following repository:

<https://github.com/philbowsher/Use-of-Python-in-regulatory-submission-related-work>

Positron provides automatic interfacing with either R or Python. Below highlights an example where a user opens a folder containing a Shiny for Python application. Positron automatically changes over to Python as highlighted below:



Many users in Positron and RStudio utilize Quarto documents when programming with open-source languages. Quarto is a multi-language, next generation literate programming tool with various features and capabilities for users when using open-source languages such as R, Python, Julia etc. Quarto enables users to create notebooks as well other output types such as dashboards, books, websites and more. Below is an example of a Quarto document that provides various code sections (chunks) for open-source languages such as Python:



Positron provides world class support for Quarto and users can read more at: <https://quarto.org/>

Quarto also supports many new tools for creating beautiful reports such as Typst. Typst is a new open-source typesetting system that creates beautiful PDF output. Examples are here:

<https://quarto.org/docs/output-formats/typst.html>

Another powerful tool for creating reports is brand.yml, which simplifies the process of applying branding guidelines to reports. brand.yml can be used in Quarto and Shiny as well as plotnine, gt, and great_tables. You can learn more about brand.yml here: <https://posit-dev.github.io/brand-yml/>

NEXT-GEN STATISTICAL COMPUTING ENVIRONMENTS (SCE)

Many pharmaceutical companies are migrating to or creating Next-Gen Statistical Computing Environments (SCE) where the focus is on open source languages for clinical reporting. A recent webinar by James Black discusses “The importance of the SCE in enabling our shift to open-source data science”:



<https://pos.it/enable-oss>

James highlights the use of Python and data science use cases in the SCEs to support new hires with experience in open source (R, Python, etc.) and use cases like Real-World Evidence. Moreover, many pharmaceutical companies use data platforms (Databricks, Snowflake, etc.), where ETL and data ingestion processes are often Python-based in accessing clinical trials and real-world data.



As pharmaceutical organizations continue to move to open source environments and blend tools like parquet, git, and Docker, they are ever focused on creating language-agnostic frameworks where programmers/developers can use numerous tools like RStudio, Jupyter, VS Code and now Positron. Positron via Posit Workbench provides session

credentials and connections to databases and datastores such as Databricks, Snowflake, Amazon S3 Cloud Storage, and DuckDB etc. as highlighted below:

Session Credentials



AWS Databricks Workspace



POSIT_SOFTWARE_PBC_DEV

Edit Credentials

Cluster Options

Resource Profile

Small

CPU(s)1Memory (GB)1.95

Image

rstudio-workbench:ubuntu2204-2024.12.0--685cadf (default)

Edit

☒ Join session when ready☐ Notify when ready

CancelStart Session


Moreover, users in Positron can use the Connections Pane for connecting to databases (<https://positron.posit.co/connections-pane.html>).

This paper will highlight the role of Positron in such an environment!


UNDERSTANDING POSITRON IN POSIT WORKBENCH

The Positron IDE was made available for Public Beta by Posit PBC in June 2024. It was first made available for desktop users and now is also available as a preview feature in Posit Workbench 2024.12.1 and later. Posit Workbench is a Server based environment that provides various IDEs as seen below:


New Session




Jupyter Notebook




JupyterLab



Positron Pro



RStudio Pro





VS Code



Session Name

Phil's Positron

Session Credentials



Posit AWS



POSIT_SOFTWARE_PBC_DEV

Edit Credentials

Cluster Options

When selecting a memory amount, use at least 4 GB

Resource Profile

Large

CPU(s)4Memory (GB)7.81

Image

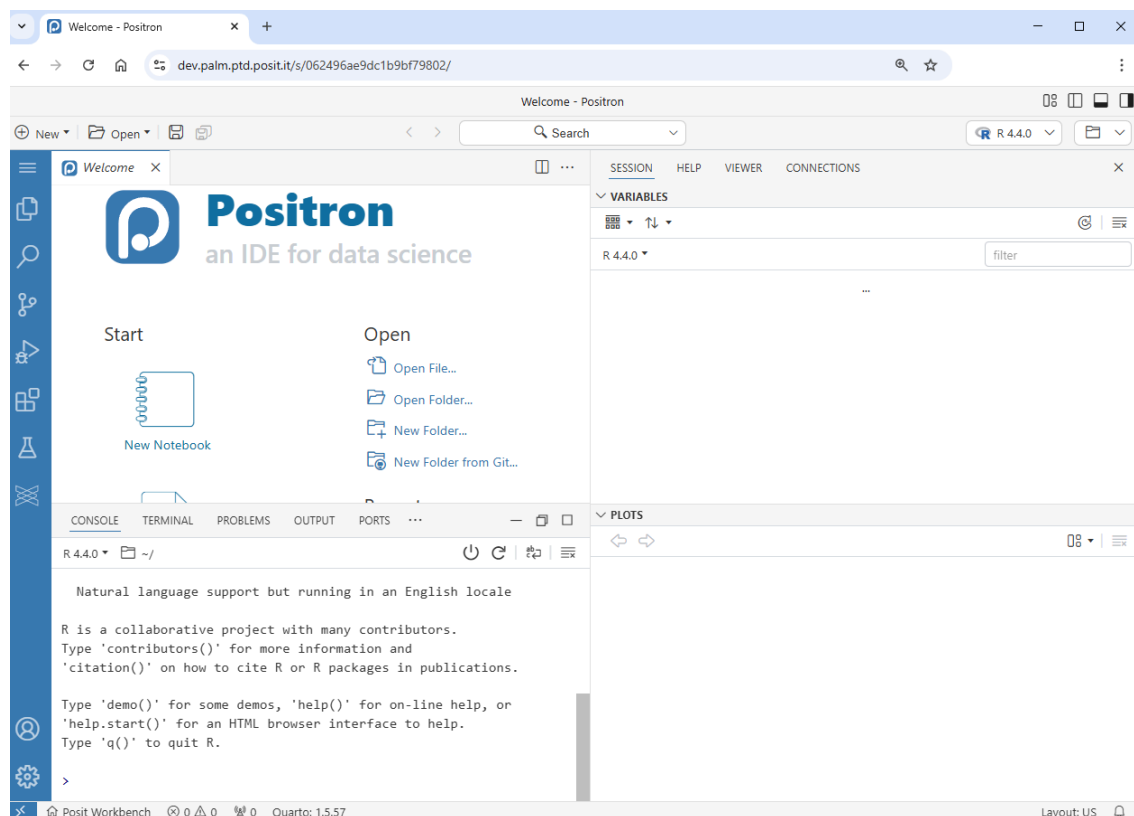
rstudio-workbench-preview:dev-jammy-2024.12.1-563.pro2 (default)

Edit

☒ Join session when ready☐ Notify when ready

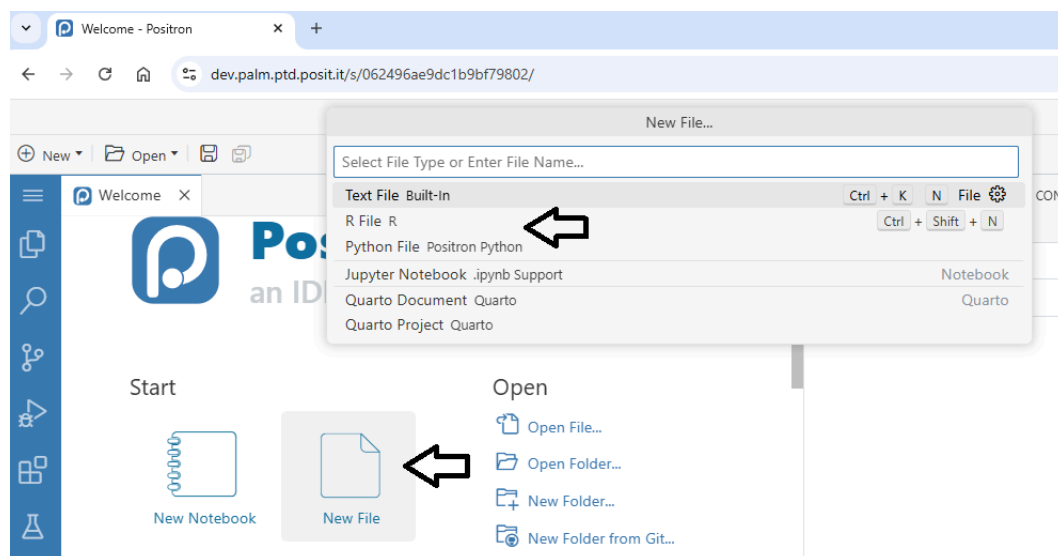
CancelStart Session

After selecting Positron in the list of IDEs, a session will start. The session will be backed by a Docker container in a Kubernetes-based orchestration system as shown here:

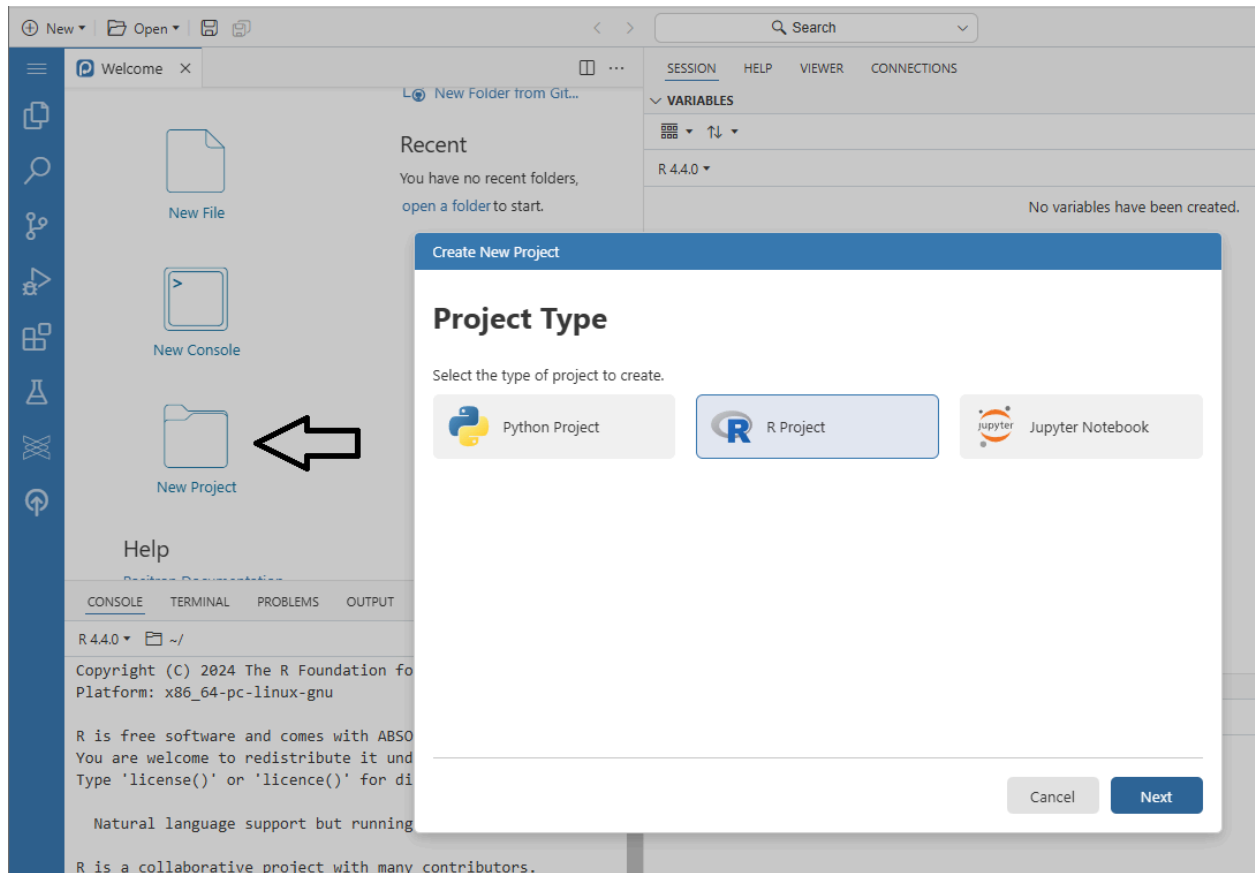


As noted, Positron runs via a server-based infrastructure within the pharmaceutical organization. Therefore, the compute is in the server or cloud based environment (AWS, Azure etc.) managed by the organization. This is demonstrated by the URL in the image above, as opposed to running the software on a desktop or laptop locally.

To open a R script, the user will select “Open File...” and pick R file as seen below:

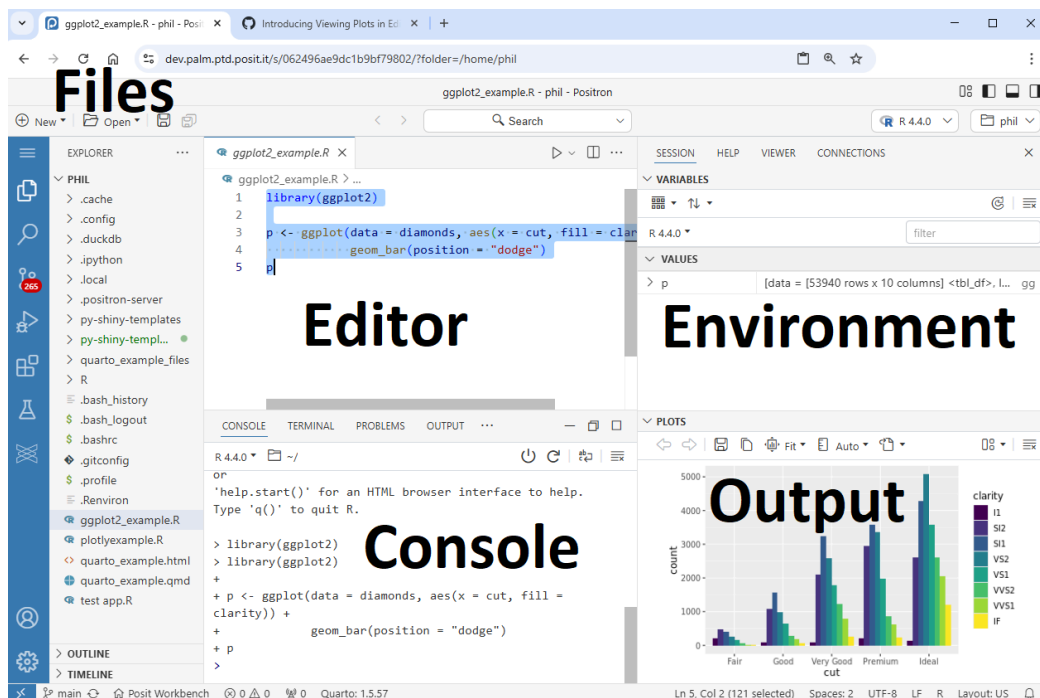


Like RStudio, Positron offers a project-oriented workflow where users can initialize version control or run git init via a terminal. To start a new project, select New Project as shown below:



4-PANE DATA SCIENCE

Posit popularized the 4-pane data science experience via the RStudio IDE. Users familiar with RStudio often miss this classic layout when using VS Code for Python or R. Positron introduces this layout automatically as seen below:



One difference is that the files are via the Explorer which is more traditional to the VS Code experience. Positron also offers an advanced Data Explorer (<https://positron.posit.co/data-explorer.html>) for code-first exploration of data. R users can run the familiar View() and review data as below, with included features for exploring and filtering data:

The screenshot shows the Positron Data Explorer interface. At the top, there is a filter bar with the expression `year = 2013` and `month >= 2`. A dropdown menu is open, showing options: `+ Add Filter`, `Hide Filters`, and `Clear Filters`. Below the filter bar, there are histograms for `dep_time`, `sched_dep_ti...`, `dep_delay`, and `arr_time`. To the right, a table displays data with columns `year`, `month`, and `day`. The status bar at the bottom indicates `Showing 309,772 rows (91.98% of 336,776 total) 19 columns`.

1. Add, Show/Hide existing filters, or Clear Filters
2. A `+` button to quickly add a new filter
3. The status bar at the bottom of the Data Explorer also displays the percentage and number of remaining rows relative to the original total after applying a filter

POLYGLOT EXAMPLE: SHINY FOR PYTHON

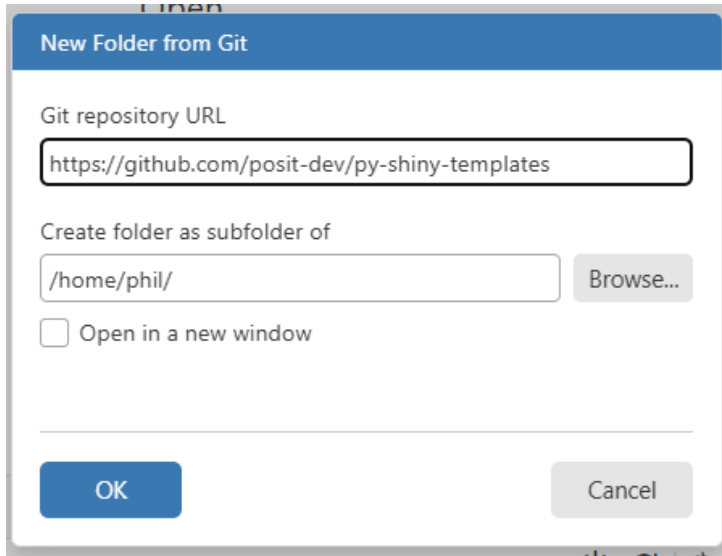
Below is an example of creating a Shiny for Python application in Positron. This example will focus on using Positron, rather than explaining the Python code to make the app. This example would be similar when creating similar output in Positron with Python for Dash, Flask, Streamlit or Bokeh. Positron supports the many Python app frameworks such as Dash, FastAPI, Flask, Gradio, Shiny and Streamlit.

Posit maintains some Shiny for Python templates at: <https://github.com/posit-dev/py-shiny-templates>

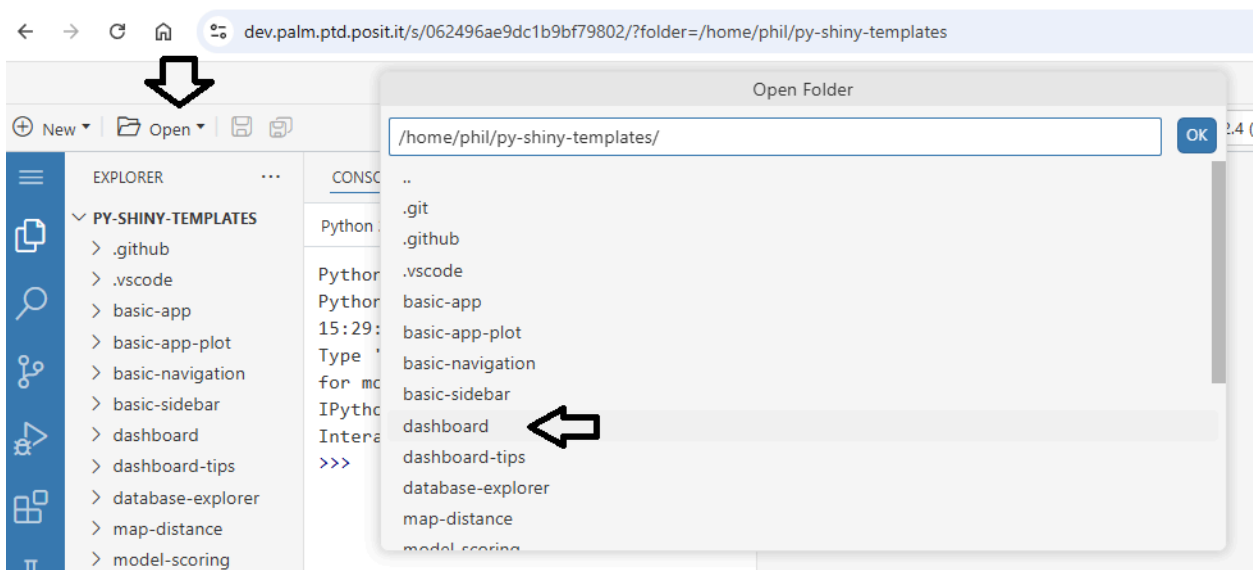
The explanation below is for an application called the Penguins Dashboard. The first step is to clone the code from Github via:

The screenshot shows the Positron IDE interface. The `Welcome` tab is active. The `Start` panel on the left shows options: `New Notebook`, `New File`, `New Console`, and `New Project`. The `Open` panel on the right shows options: `Open File...`, `Open Folder...`, `New Folder...`, and `New Folder from Git...`. A black arrow points to the `New Folder from Git...` option. Below the `Open` panel, the `Recent` section shows the following paths: `phil /home` and `py-shiny-templates ~`.

This will open the New Folder dialog. Add the Github link to the Shiny for Python templates and click OK.

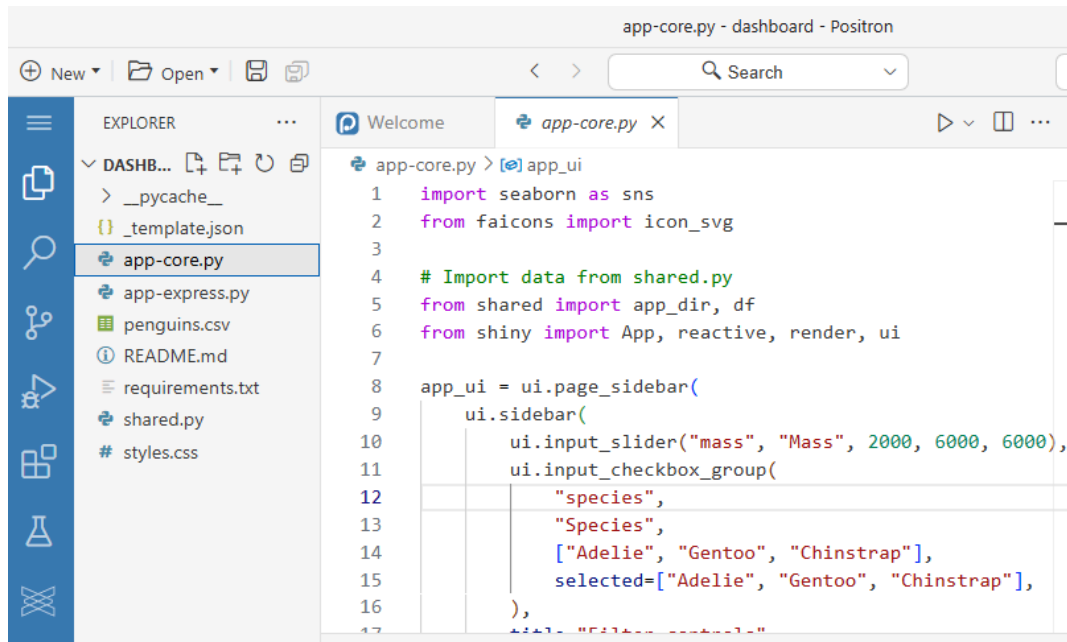


Now inside of Positron we will see a folder containing the Shiny for Python Template examples. Within this newly cloned folder, we will open the Dashboard example:

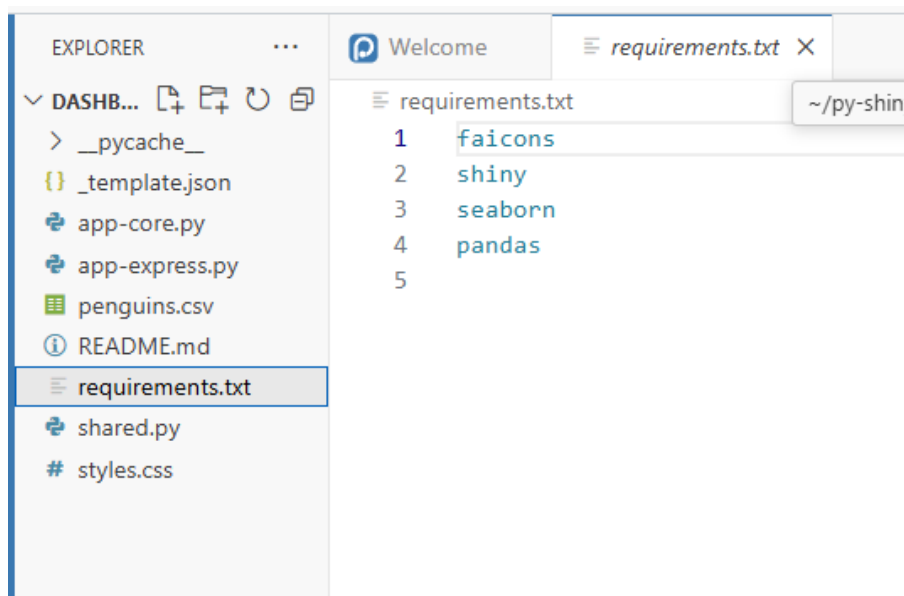


After the Dashboard folder opens, Positron will refresh and we will now see the files for this application. Also note that Positron has detected that Python is being used and the Console and Language has switched from R to Python automatically.

For this example, we will open the `app-core.py` file in Positron. There is also an `app-express.py` file. Shiny for Python has 2 ways to make applications, Shiny Express or Shiny Core. Shiny Express is designed to make it easier to get started with Shiny, and to create simple apps with a boilerplate. Shiny Core is the traditional way to make Shiny for Python apps.



Before we can run the Application in Positron, we need to set up the Python environment. In addition to various Shiny for Python functions, the Dashboard utilizes the Python packages, faicons, pandas and seaborn. These Python packages can be installed in Positron via a *requirements.txt* file often bundled within the App folder. This can be accomplished by install the Python packages in the requirements.text file:



To do this, we need to run the following command in the Python Console:

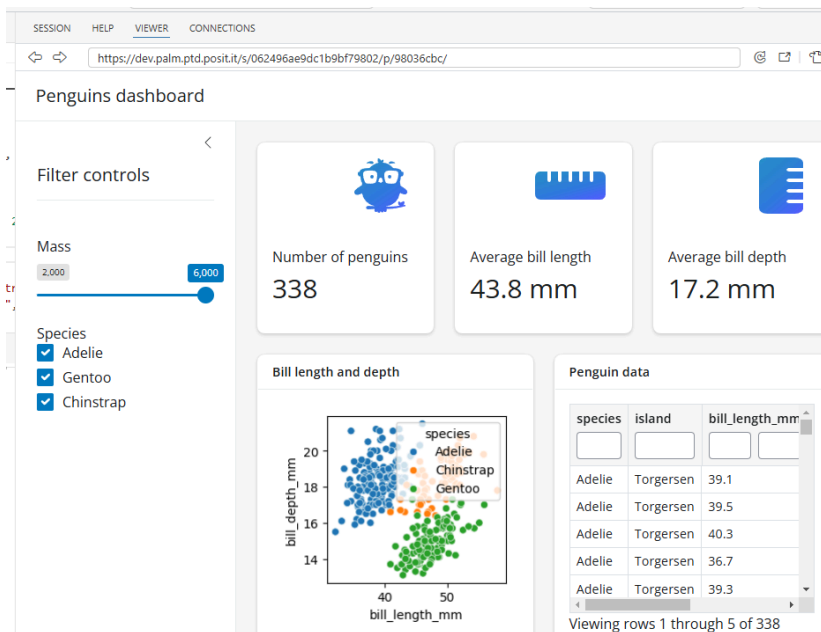
```
pip install -r requirements.txt
```

After the Python packages are installed, the Shiny app can be generated by clicking the “Run” button. It is important to note that no additional extensions were required to add this functionality.

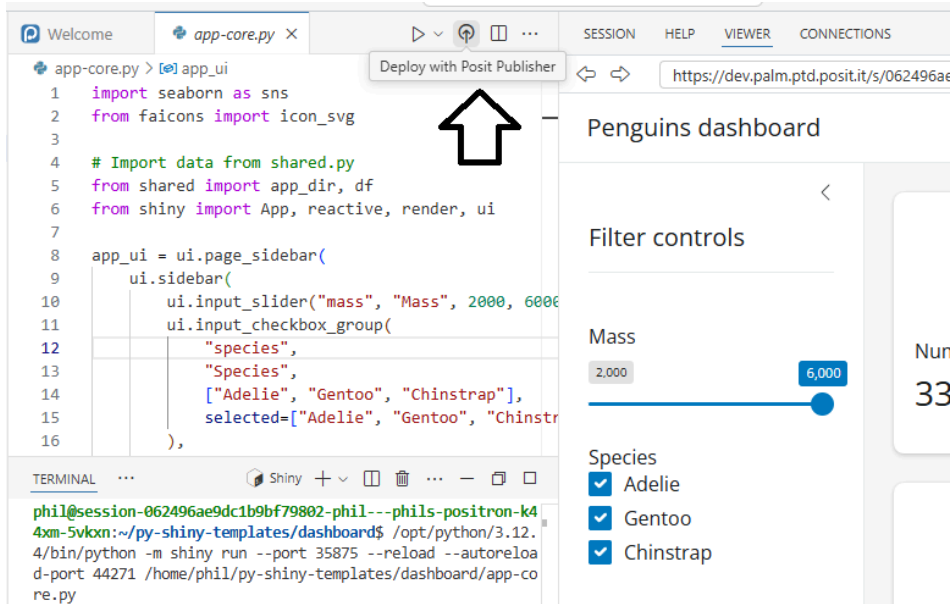
```
app-core.py X
app-core.py > [app-ui]
1 import seaborn as sns
2 from faicons import icon_svg
3
4 # Import data from shared.py
5 from shared import app_dir, df
6 from shiny import App, reactive, render, ui
7
8 app_ui = ui.page_sidebar(
9     ui.sidebar(
10         ui.input_slider("mass", "Mass", 2000, 6000, 6000),
11         ui.input_checkbox_group(
12             "species",
13             "Species",
14             ["Adelie", "Gentoo", "Chinstrap"],
15             selected=["Adelie", "Gentoo", "Chinstrap"],
16         ),
17         title="Filter controls",
18     ),
19     ui.layout_column_wrap(
20         ui.value box(
```



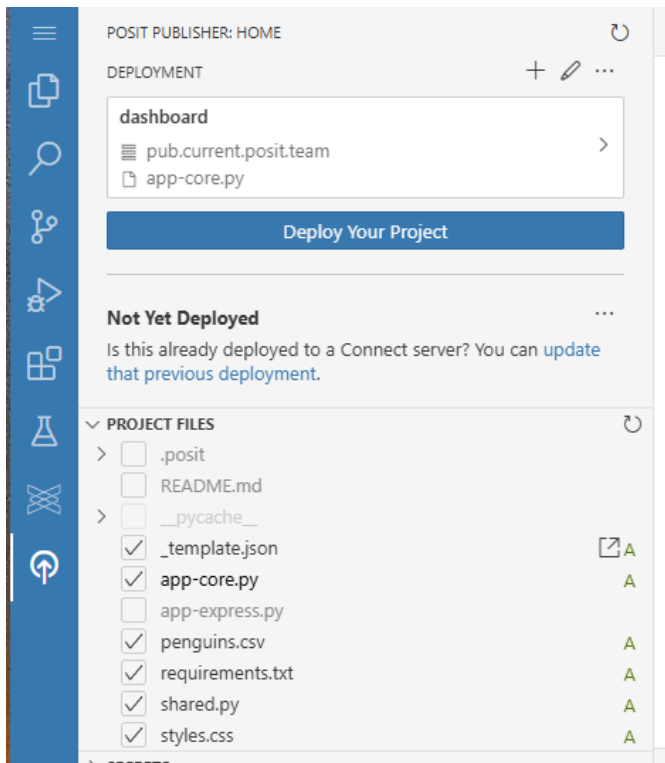
A preview of the Shiny application will show up in Positron under the Viewer pane:



Now that the application is running and working, users often want to publish the content to a Production Server to share with other colleagues. One example is Posit Connect, where we can publish the Python content. Pushing the “Publish” button below will bundle the content and requirements and ship to a Posit Connect environment as seen below:

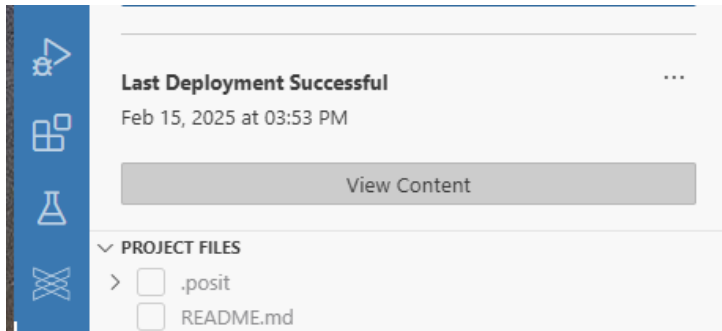


After clicking the Publish button and adding in the name, server and other information, the content will be ready to be deployed to the Posit Connect environment:

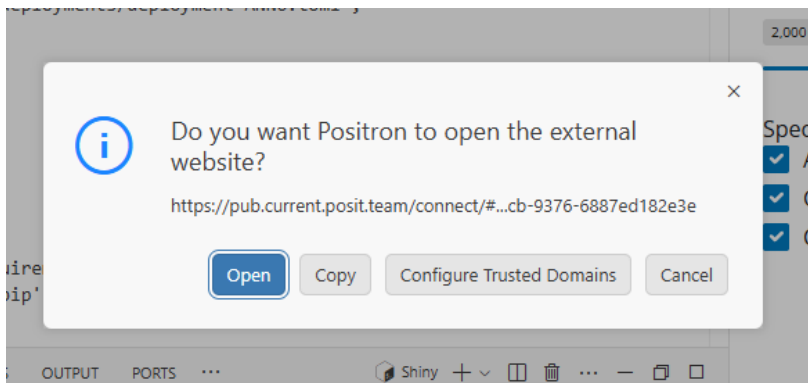


After deploying the Shiny app, Positron will communicate with the Posit Connect server to share the requirements needed to house the content. To view the content:

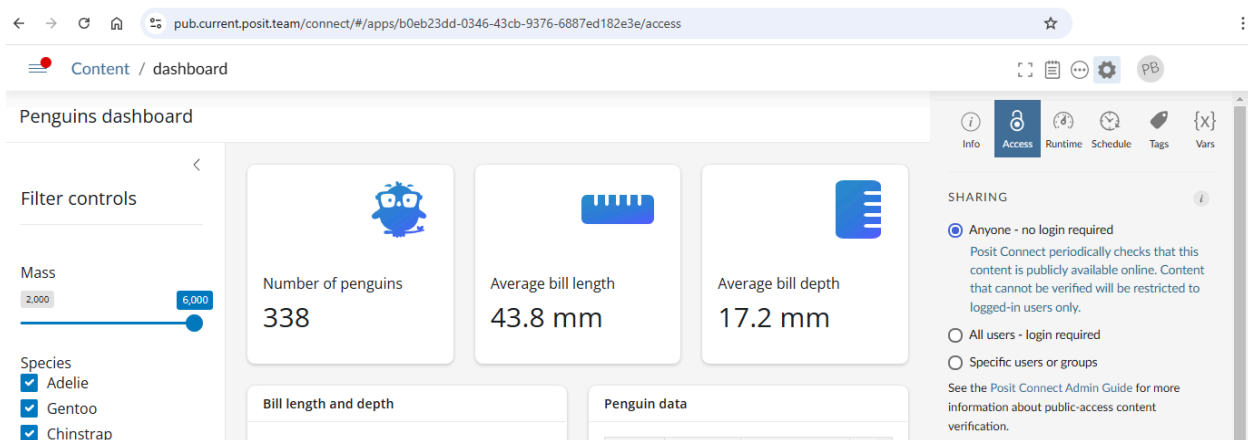
1. Click View Content



2. Click Open



3. View Content in Posit Connect



GenAI Options in Positron

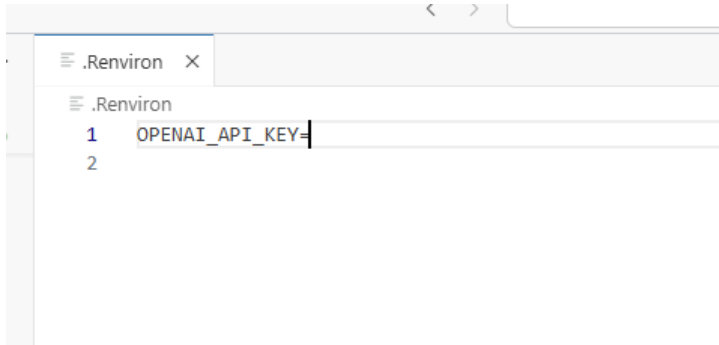
Positron supports Generative AI (GenAI) capabilities in various ways. More GenAI UI and capabilities are planned to be built directly within Positron for the future. Below will explore 2 current ways to use modern GenAI capabilities within Positron via tools created by Posit PBC. Positron is also compatible with GenAI extensions on OpenVSX such as Google Gemini, AWS Q, and Continue.dev.

You can read about other Posit PBC GenAI capabilities here: <https://www.tidyverse.org/blog/2025/01/experiments-llm/>

A. ellmer for R

ellmer provides an interface to large language models (LLM) from R. It supports many LLM providers and provides rich capabilities to use within Positron that will be explored below with ChatGPT. The ellmer main page is: <https://github.com/tidyverse/ellmer>

Below will highlight how to use ellmer and ChatGPT. First, edit the .Renviron file by adding a OPENAI_API_KEY key as shown below:



Be sure to close the file and restart the R interpreter. If needed, you can install ellmer with:

```
install.packages("ellmer")
```

To start using ellmer, create a new chat object as shown below:

```
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.

ellmer can be run in various ways such as chatting directly in the R console with `live_console(chat)` or browser with `live_console(chat)`. Another option is to make a programmatic call such as:

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

Now let's use ellmer and ChatGPT to create a Shiny app. First, run the following code in an .R file.

```
result <- chat$chat("create a shiny app that uses R, ggplot2 and plotly. Also use the pharmaverseadam  
adam data at https://github.com/pharmaverse/pharmaverseadam and create an adverse events plot ")
result
```

After the code runs, it will generate the code in the console like:

```

CONSOLE  TERMINAL  PROBLEMS  5  OUTPUT  PORTS  DEBUG CONSOLE

R 4.4.0 ~/  

Here is the revised Shiny app code:  

````r  

library(shiny)

library(ggplot2)

library(plotly)

library(dplyr)

library(pharmaverseadam)

Load the ADAM adverse events dataset from the pharmaverseadam package

adae <- adae

Data preprocessing: Summarizing the adverse events

ae_summary <- adae %>%

 filter(!is.na(AEBODSYS)) %>%

 group_by(AEBODSYS) %>%

 summarise(Count = n(), .groups = "drop")

Define UI

ui <- fluidPage(

 titlePanel("Adverse Events Plot"),

 plotlyOutput("aePlot")

)

Define Server

server <- function(input, output) {

 output$aePlot <- renderPlotly({

 gg <- ggplot(ae_summary, aes(x = reorder(AEBODSYS, -Count), y = Count)) +

 geom_bar(stat = "identity", fill = "steelblue") +

 labs(x = "Adverse Event Body System", y = "Count of Adverse Events") +

 theme_minimal() +

 coord_flip()

 ggplotly(gg)

 })

}

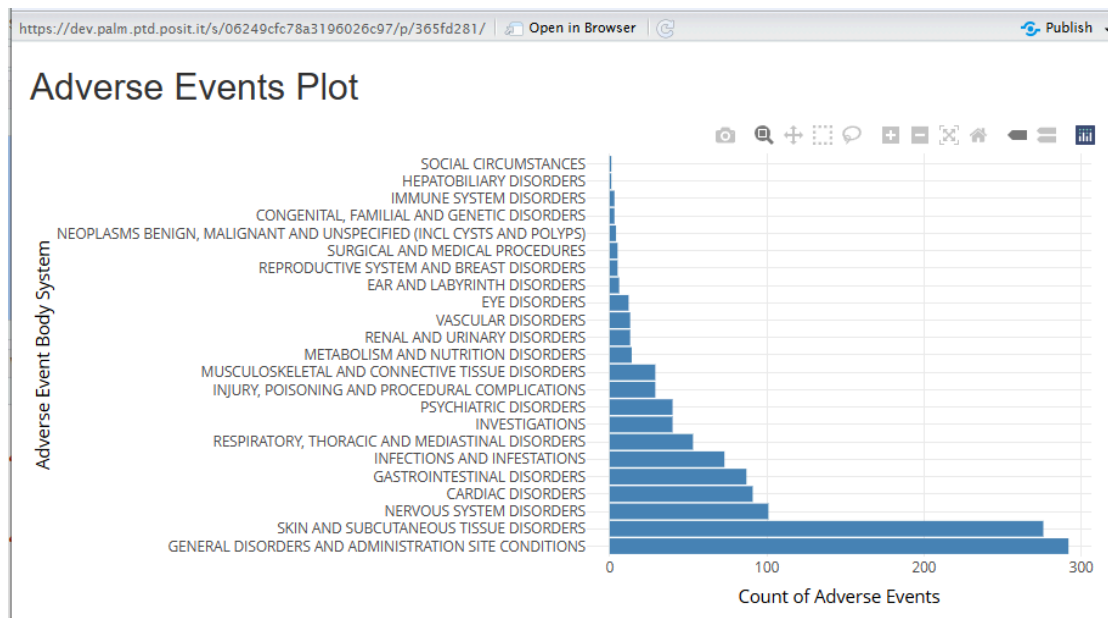
Run the app

shinyApp(ui, server)

````

```

Copy this code from the console to a .R file within a folder. Run the code to see the Shiny app!



It is important to note that the first few times I asked for the code, ChatGPT hallucinated and made calls to data sets that did not exist. This was circumvented by prompting again and being explicit on what code to use to pull the data.

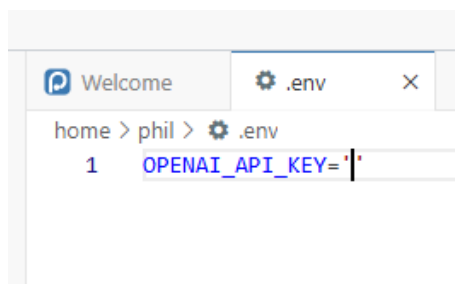
B. chatlas for Python

For Python users, chatlas provides an interface to large language model (LLM) providers. chatlas can be installed in Positron via:

```
pip install -U chatlas
```

Like ellmer, chatlas provides several different ways to engage with it such as interactively or programmatically. It is important to remember that chatlas is for Python so users may need to switch to the Python interpreter in the top right corner if not automatic.

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`. In the file, add the following:



Users then use the python-dotenv package to load them into the environment.

```
pip install python-dotenv
```

Save the file, restart the Python interpreter and refresh the Positron session. Now install the OpenAI Python library via the Python console, which provides access to the OpenAI REST API.

```
pip install openai
```

Now let's use chatlas and ChatGPT to create a Shiny app. First, run the following code in an `.py` file:

```
import os
from chatlas import ChatOpenAI
from dotenv import load_dotenv
load_dotenv()
chat = ChatOpenAI(
    model = "gpt-4o",
    system_prompt = "You are a friendly but terse assistant.",
)
chat = ChatOpenAI(api_key=os.getenv("API_KEY"))
chat.console()
```

In the Python console, type:

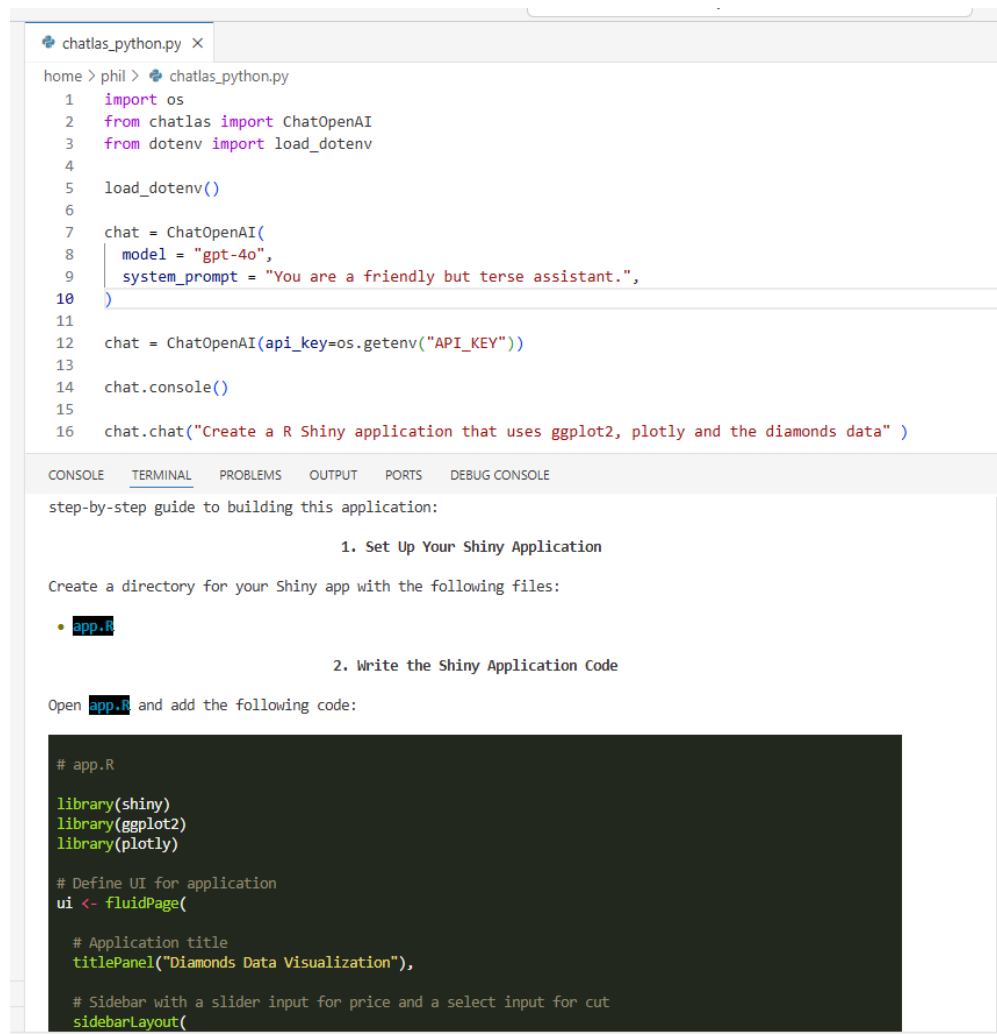
```
"Create a R Shiny application that uses ggplot2, plotly and the diamonds data"
```

This prompt could be further customized for pharma with an example like:

"Create a shiny for python application that analyzes the openfda api data and creates a great_tables table and a plotnine graphic"

Alternatively, you can use a programmatic approach with the `.chat()` method as in:

`chat.chat("Create a R Shiny application that uses ggplot2, plotly and the diamonds data")`



The screenshot shows a Positron IDE window with a tab titled `chatlas_python.py`. The editor displays a Python script that initializes a `ChatOpenAI` object and calls the `.chat()` method with a specific prompt. Below the editor, the `TERMINAL` tab is active, showing a step-by-step guide to building a Shiny application. The guide includes two steps: '1. Set Up Your Shiny Application' and '2. Write the Shiny Application Code'. Step 2 provides a code snippet for `app.R` that sets up a Shiny application with a title panel and a sidebar layout.

```
home > phil > chatlas_python.py
1 import os
2 from chatlas import ChatOpenAI
3 from dotenv import load_dotenv
4
5 load_dotenv()
6
7 chat = ChatOpenAI(
8     model = "gpt-4o",
9     system_prompt = "You are a friendly but terse assistant.",
10 )
11
12 chat = ChatOpenAI(api_key=os.getenv("API_KEY"))
13
14 chat.console()
15
16 chat.chat("Create a R Shiny application that uses ggplot2, plotly and the diamonds data" )
```

CONSOLE **TERMINAL** PROBLEMS OUTPUT PORTS DEBUG CONSOLE

step-by-step guide to building this application:

1. Set Up Your Shiny Application

Create a directory for your Shiny app with the following files:

- `app.R`

2. Write the Shiny Application Code

Open `app.R` and add the following code:

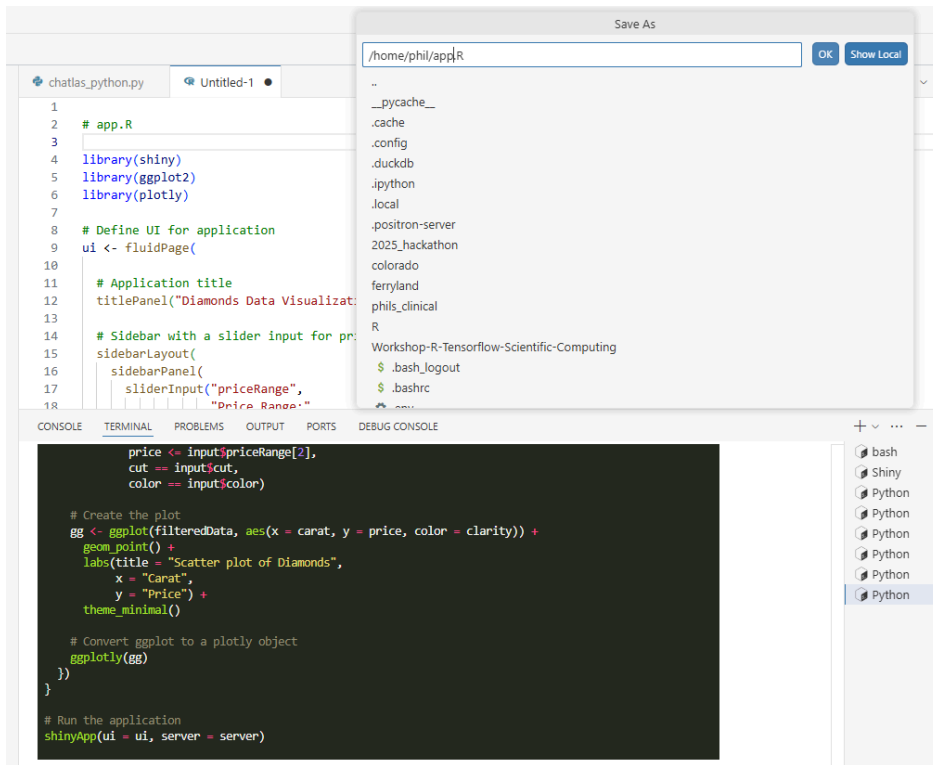
```
# app.R
library(shiny)
library(ggplot2)
library(plotly)

# Define UI for application
ui <- fluidPage(

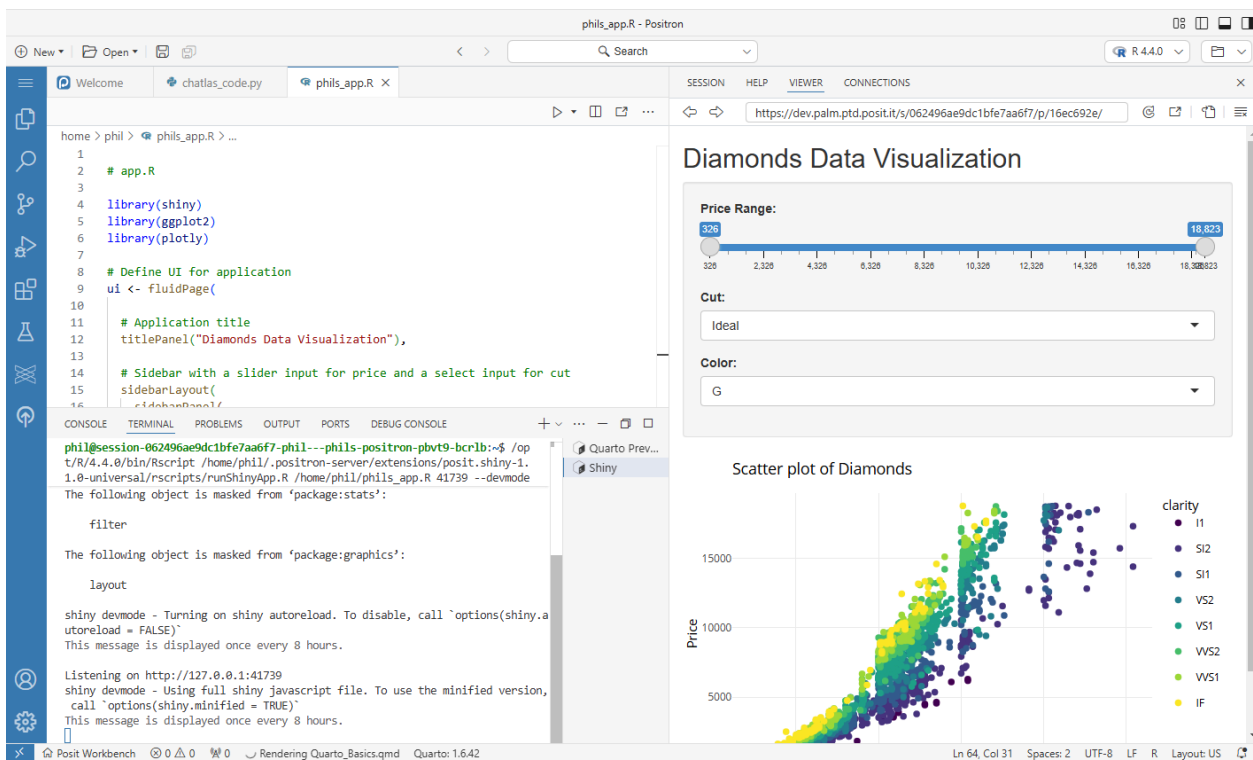
  # Application title
  titlePanel("Diamonds Data Visualization"),

  # Sidebar with a slider input for price and a select input for cut
  sidebarLayout(
```

Copy the Shiny R code into an `app.R` file and run it inside of Positron like:



This code then generates the following app:



CONCLUSION & CONTACT

The information above highlights the exciting new IDE, Positron and example use cases. Positron and open source

languages like R and Python can help modernize clinical processes with innovative new features and capabilities.

Phil Bowsher

phil@posit.co

<https://github.com/philbowsher>