PharmaSUG 2022 - Paper AP-188

Pins for Sharing Clinical R Assets

Phil Bowsher, RStudio PBC;
Ryan Johnson, RStudio PBC

Abstract

The pins package helps statistical programmers publish and share data sets, models, and other assets across the organization. Programmers can pin objects to a variety of “boards” and others can download and access the data/objects from a central location. The goal of this paper is to introduce statistical programmers to the pins package and how to share data sets across the organizations in a more reproducible way than legacy methods like emailing data. The main pins site for reference is here:

https://pins.rstudio.com/

Introduction

The pins package was created in early 2019 with the primary purpose of publishing and sharing R objects. Often workflows are restricted to local data that collaborators do not have access to. Other times, the data has grown stale and needs to be refreshed. In other situations, data are in complex relational databases that can be tricky to query.

The above examples complicate data access across projects and teams. Moreover, managing access control can be difficult in situations involving sensitive data. The pins package can help remedy these situations by providing an accessible place to access data for various workflows. Moreover, ***processes such as ETL (Extract, Transform, Load) jobs*** can benefit from pins integration to ensure data is refreshed on an interval specified by the data owner.

This paper is for statistical programmers that are building clinical routines and have wondered how best to approach datasets in workflows.

Understanding Pins and Boards

Pins can be used to publish R objects in a variety of formats (RDS, CSV, arrow/feather, JSON, etc.). Small data sets are the most commonly pinned object, such as ephemeral data or reference tables, not suited for databases, but still require a physical “home” that workflows can point to. These homes are known as “boards.” Boards have a variety of storage backends which are detailed here: https://pins.rstudio.com/reference/index.html#boards

Example Workflow

A common workflow is as follows:

An R developer has processed raw data resulting in a polished dataset that is ready to be accessed by other workflows (eg., a clinical table pipeline or Shiny application). Here is a sample pins workflow:

Load pins package
This will make the functions of the pins package available to users.

Create the board
This will create a location to store the object.

Write data to the board as a pin
This process will transfer the object from RStudio to your board.

Then others at the organization can access the pin like this:

Load pins package
This will make the functions of the pins package available to users.

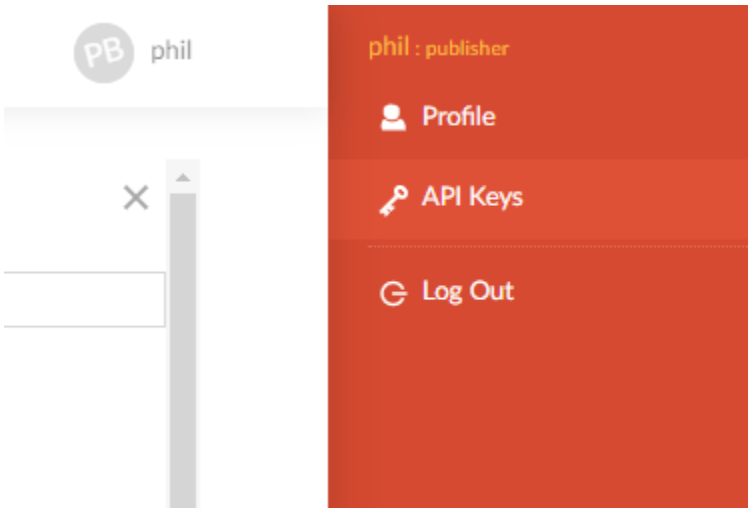
Identify the board
This will tell R the location and name of the board we want to use that has the pin.

Read the shared data
This will bring the pin into the RStudio connections tab. Users can also create their own datasets or objects as a copy for their own work.

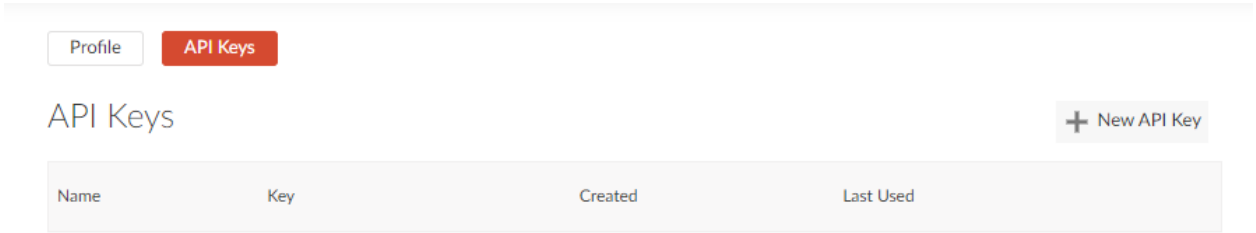
Understanding API Keys and Access Controls with RStudio Connect

RStudio Connect can serve as a board for pins and provide an additional layer of access control. For users that wish to access a secure pin on RStudio Connect, a user specific API key will need to be generated as described below. Please ensure you treat API Keys like a password. API keys enable you to send requests to systems as if you were logged in manually. See the API Keys documentation for more details:
https://docs.rstudio.com/connect/user/api-keys/

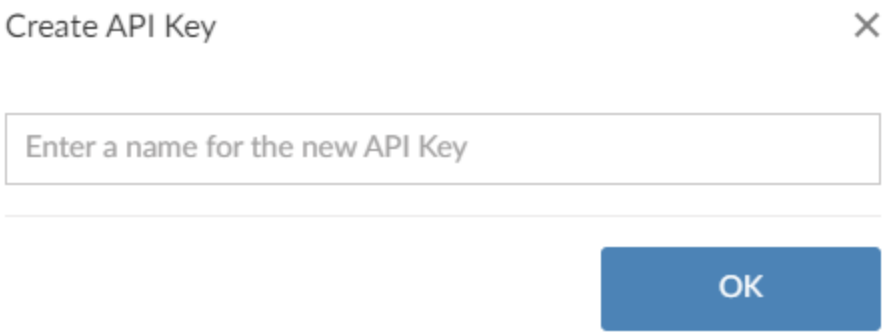
Click on your name in the top right corner of RStudio Connect. Then click on API Keys:



Then click on New API Key:



Then create a name for your key.



Make sure you save your key in a safe place.

An important package to manage API keys is the usethis R package:

https://usethis.r-lib.org/

It is advised to place your API token as an environment variable in your `.Renviron` file. The usethis package makes it easy to edit this file for this purpose.

The code would look like this:

usethis::edit_r_environ()

Then enter in the file and save:

CONNECT_API_KEY="your-api-key"

CONNECT_SERVER="`https://rstudio-connect-server`"

These API keys allow users to programmatically access content on RStudio Connect and use the Connect Server API.

Below we will show a real workflow in R.

Adverse Events Example

The FDA maintains an API that houses a number of high-value, high priority and scalable structured datasets, including adverse events, drug product labeling, and recall enforcement reports.

https://open.fda.gov/

The R package, openfda, provides convenient access to the OpenFDA API. Data in the API are often changing as new data are sent from hospitals, doctors, and other organizations.

Below we will work to create a reproducible workflow that creates a master dataset that is pinned to RStudio Connect. Then we will create a Shiny app and dashboard that will use the pinned data.

All of the examples below are available here:

https://github.com/philbowsher/RMD-Shiny-session-2020-4-21/tree/master/RStudio-Connect-session-2019-12-06/shiny-days-master/Pins

Creating the Living Dataset and ETL Workflow

Below we will create the ETL job that we can run weekly to check for new data. We will schedule the ETL job to update our pinned data set each week so that all other processes and artifacts that depend on the data will always have the updated results.

For this example, we will use a R Notebook. It is important to either use a R Notebook, R Markdown or Quarto file as we will want the report to be executable by our production system, in this example it will be RStudio Connect.

https://raw.githubusercontent.com/philbowsher/RMD-Shiny-session-2020-4-21/master/RStudio-Connect-session-2019-12-06/shiny-days-master/Pins/02_building_blocks.Rmd

Our ETL job will have the following structure:

1. Load the packages
2. Create helper functions to use the openFDA API and package to query adverse events data
3. Using our openFDA helper functions, we can pull adverse event data by gender for a specific drug and wrangle as needed
4. Publish our master data set as a pin to our board
5. Create some plots to test that our workflow is working

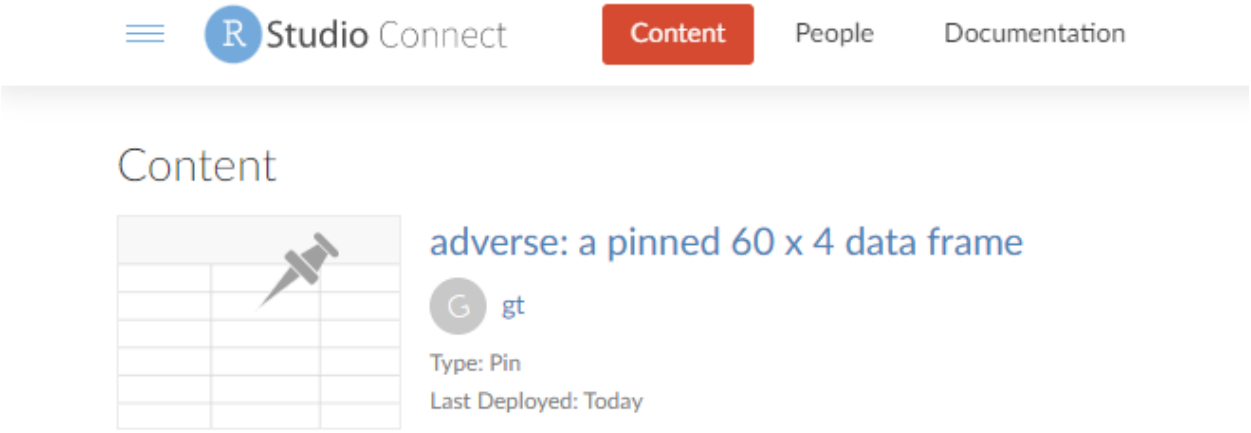
The pins code looks like this:

````
## Publish Pin
```{r}
library(pins)

# depends on CONNECT_SERVER and CONNECT_API_KEY variable
board <- board_rsconnect()

board %>% pin_write(adverse, "adverse", type = "csv")
```
````

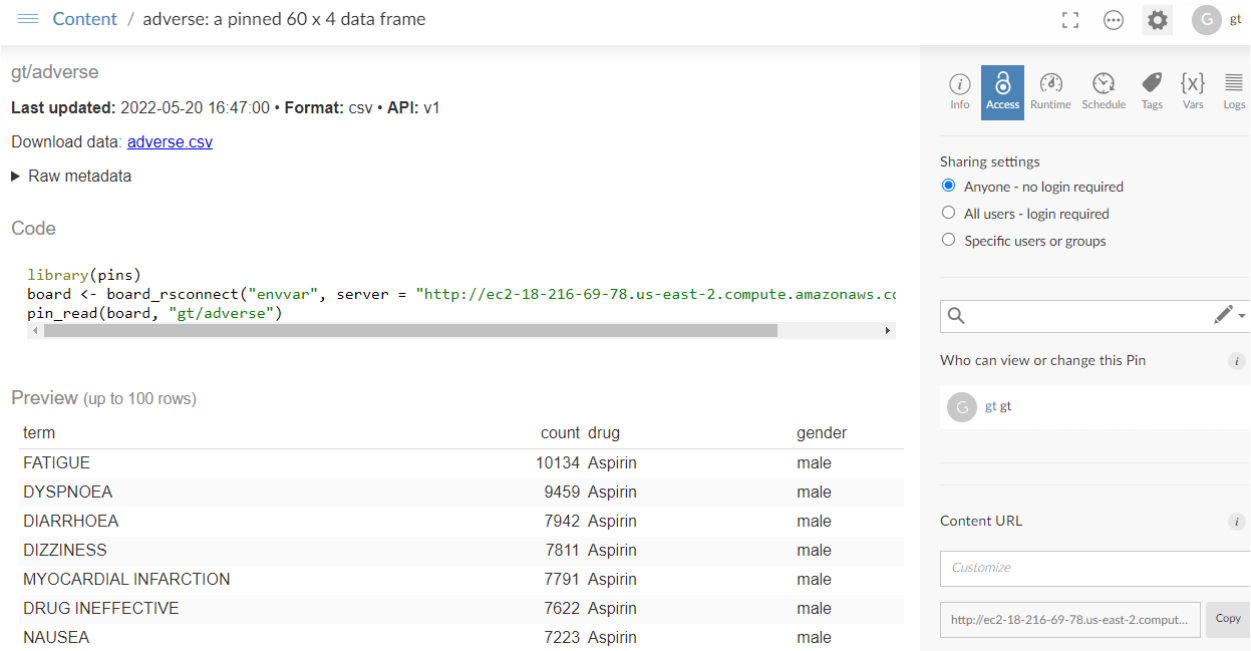
For Connecting to your board in the code above, you will need to update your code with your API key like this:

```
board <- board_rsconnect(auth = "envvar", server =
"http://ec2-18-216-69-78.us-east-2.compute.amazonaws.com/rsconnect/",key =
Sys.getenv("VETIVER_API"))
```

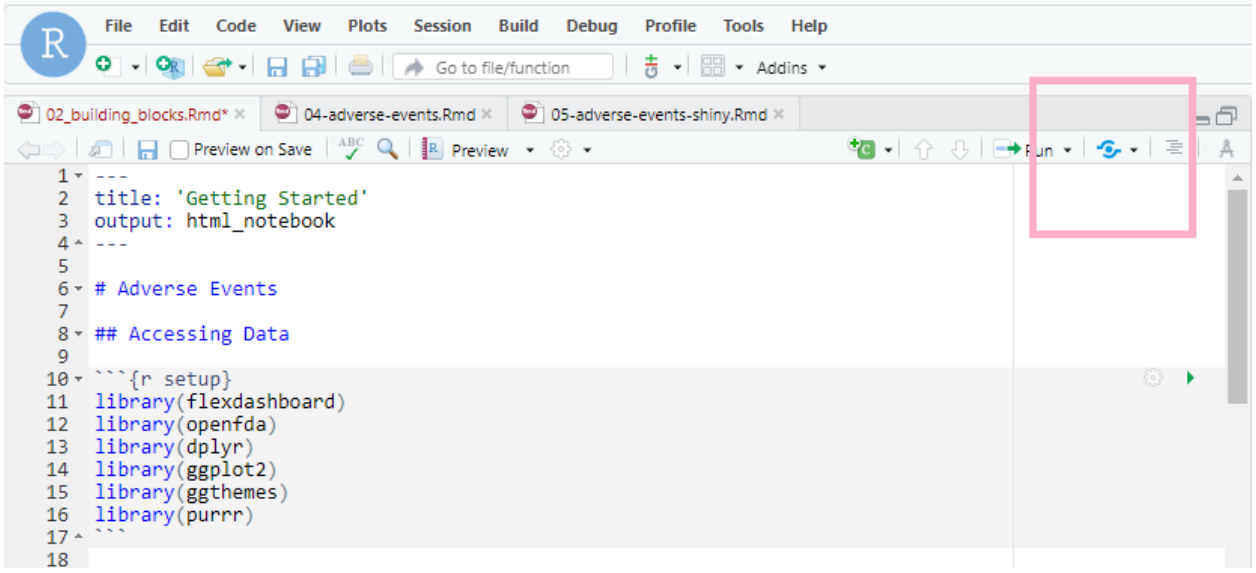
Now your pinned dataset will show up in RStudio Connect like this:



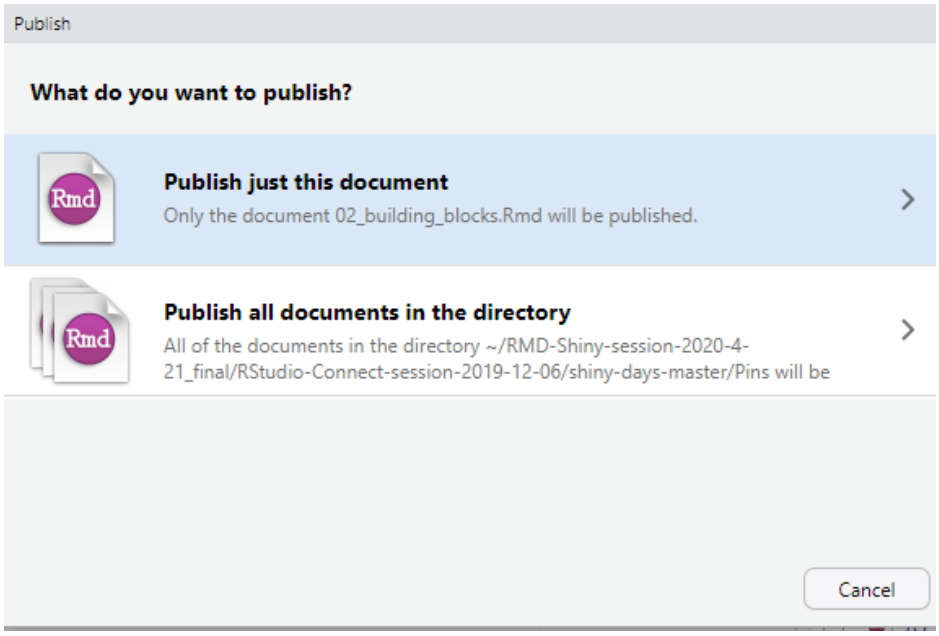
After clicking on the data set in RStudio Connect, viewers will see:



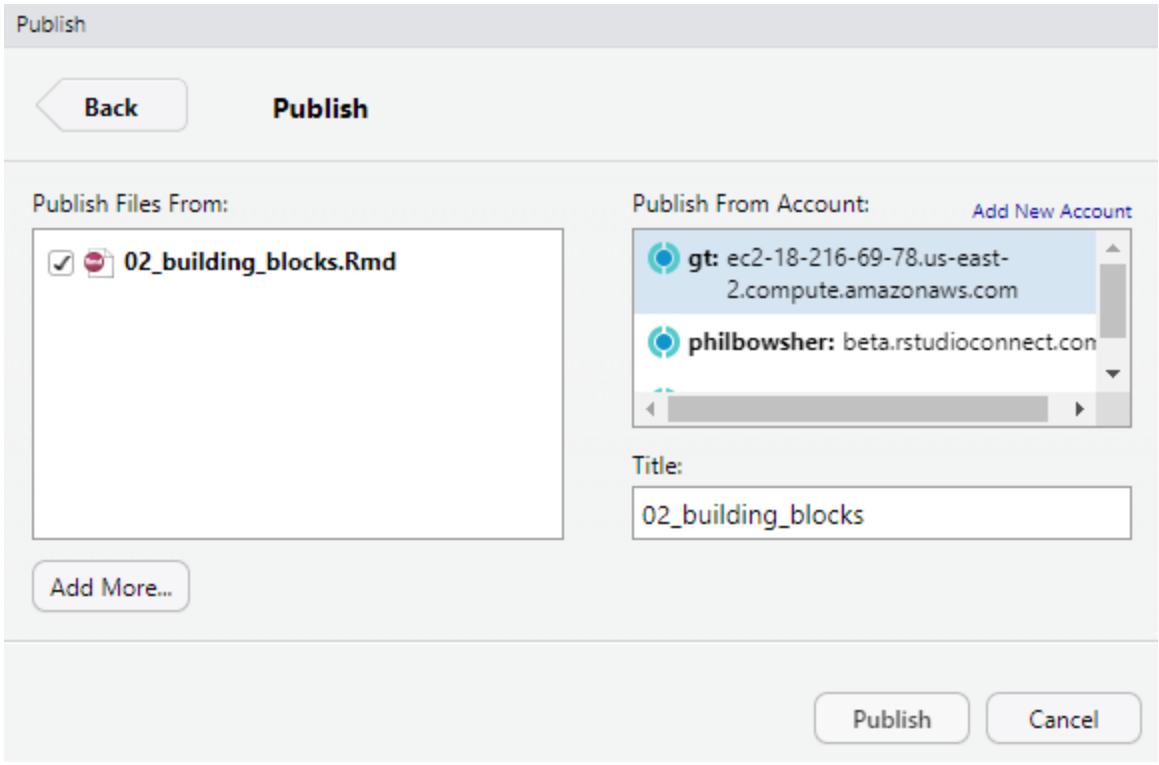
Now that the data prep is working and our data is pinned properly, we can publish the R Notebook ETL to the RStudio Connect Production Environment. This can be done by clicking the blue publish button at the top right of the file like this:



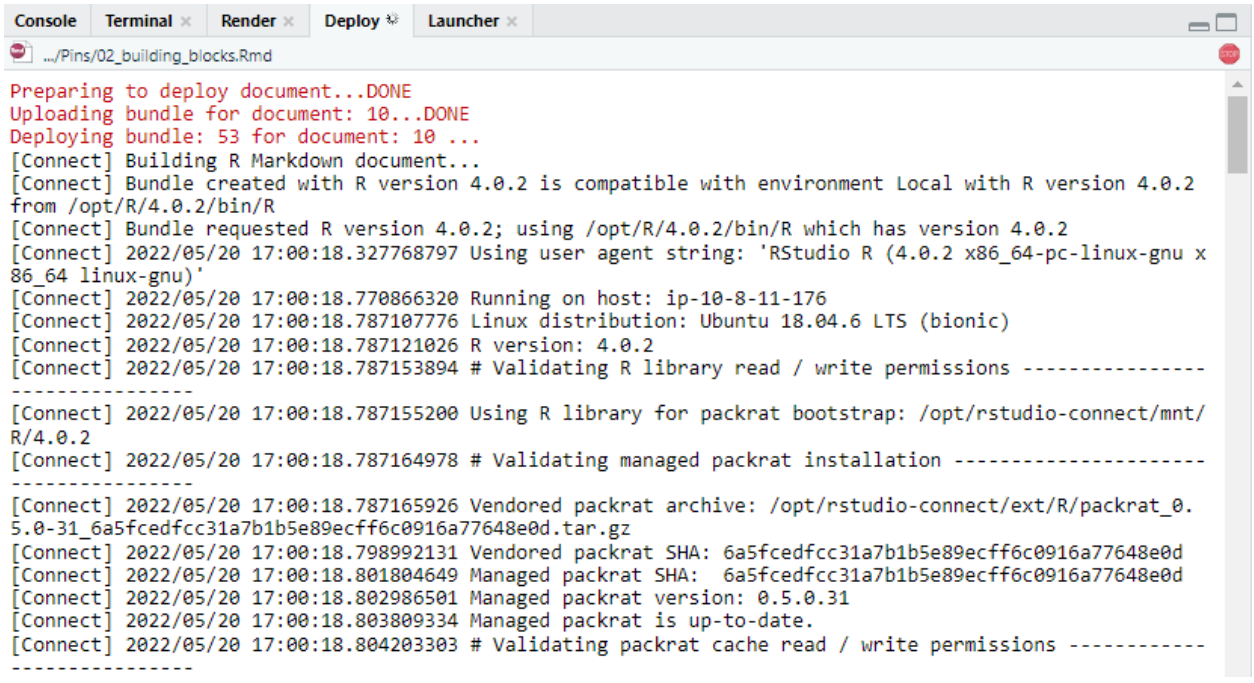
Be sure to publish the document and any other files needed. In our case, it will just be the one ETL job.



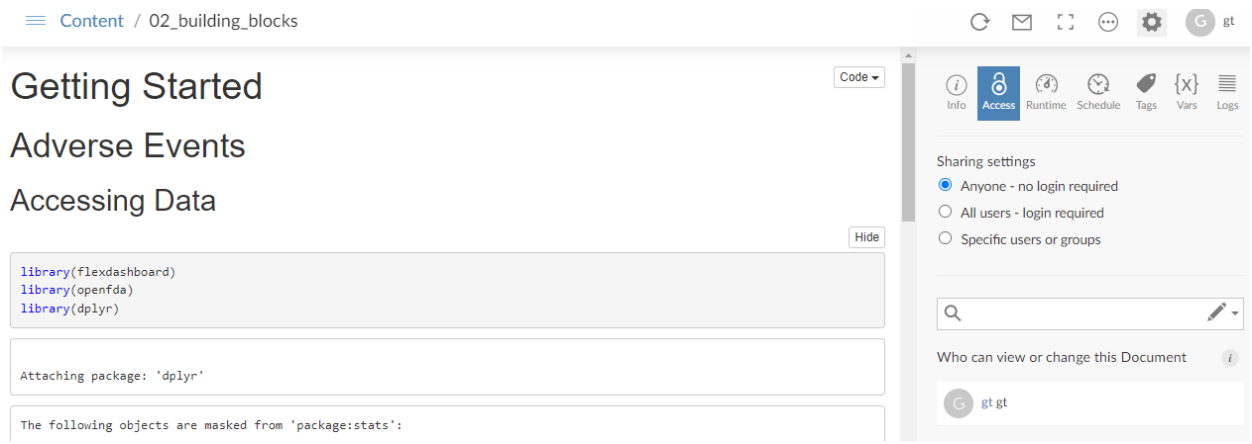
If it is your first time publishing, there will be a series of steps for authenticating into the RStudio Connect Production server. After those steps, you can publish like this:



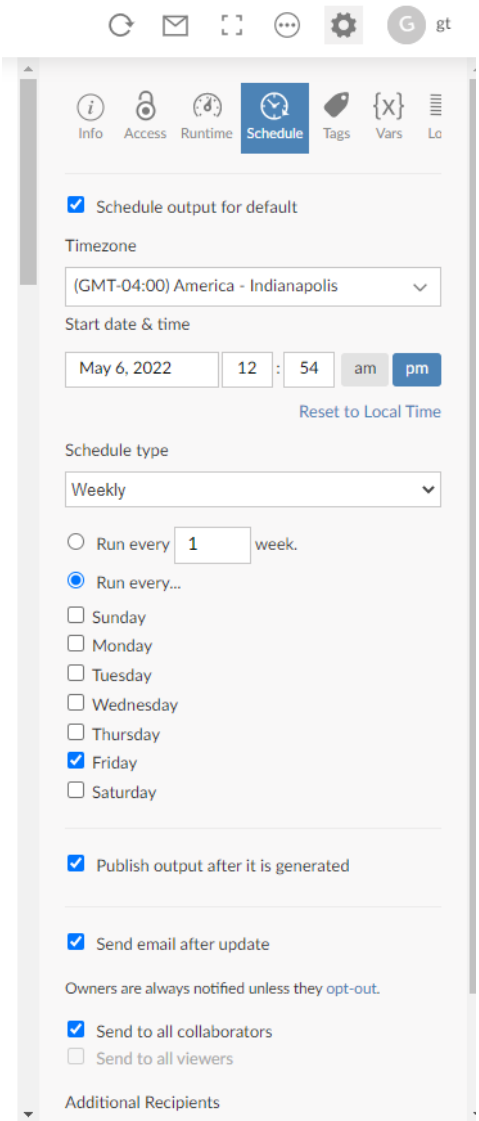
After “publish” is clicked, RStudio Workbench will communicate with RStudio Connect and pass along the required information for rebuilding a sandbox where the ETL job will live in Production. In Production, the ETL job will have the required packages and dependencies to operate as needed:



After the packages are set up, the content will show in RStudio Connect:



To the right, the user can set the required settings for the ETL job such as access and scheduling. Since we want our pinned dataset to be refreshed weekly, we can set the ETL job to run systematically to refresh our pinned data:



This process will also send an email letting us know our ETL job ran successfully.

Please note that content like our ETL job above can also be deployed with github and or CI/CD. Please read this doc for more information.

https://www.pharmasug.org/proceedings/2021/AD/PharmaSUG-2021-AD-205.pdf

Creating Artifacts and Content that Use our Pinned Data

Now that our data are pinned to RStudio Connect, we can create reports, dashboards, Shiny apps, etc. that use the data. Below we will do that.

Flexdashboard R Markdown Dashboard

Flexdashboard is a great package for creating dashboards:

https://pkgs.rstudio.com/flexdashboard/articles/examples.html

Below is a flexdashboard we created that uses our pinned data:

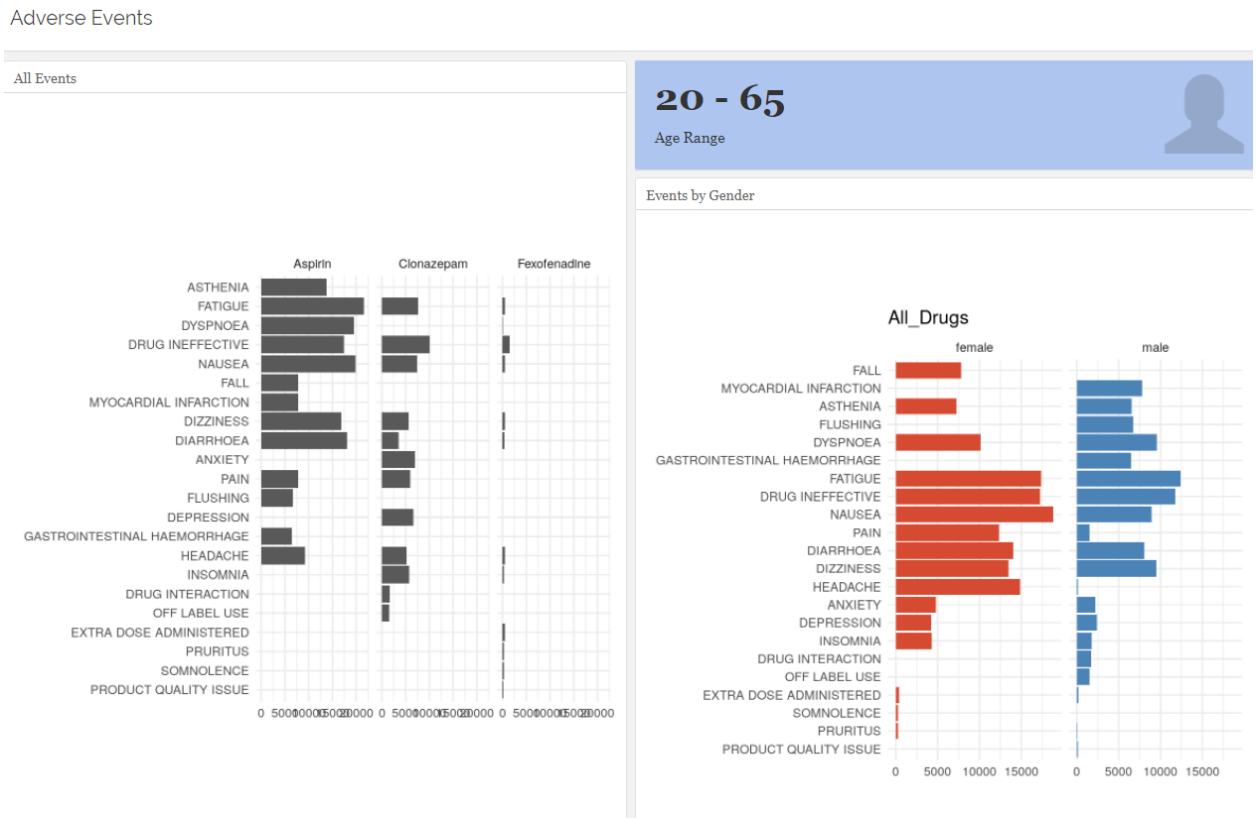
https://github.com/philbowsher/RMD-Shiny-session-2020-4-21/blob/master/RStudio-Connect-session-2019-12-06/shiny-days-master/Pins/04-adverse-events.Rmd

Now that the data are in RStudio Connect, we do not need to include the data wrangling routine in our dashboard code as it is now handled by our published ETL job. For example, the pink section blow is no longer required:

```
38
39 ```{r}
40 #this to be replaced by pins pull
41
42 age <- create_age(20,65)
43
44 # jnk <- capture.output(male <- get_adverse("1", params$drug, age))
45 # if (!is.null(male)) {
46 #  male$gender <- 'male'
47 # }
48
49 # jnk <- capture.output(female <- get_adverse("2", params$drug, age))
50 # if (!is.null(female)) {
51 #  female$gender <- 'female'
52 # }
53
54 # adverse <- rbind(male, female)
55
56 # adverse <- pin_get("Adverse", board = "rsconnect") %>% group_by(term) %>% summarise(count =
   sum(count))
57
58 library(pins)
59 board <- board_rsconnect("envvar", server =
   "http://ec2-18-216-69-78.us-east-2.compute.amazonaws.com/rsconnect", key =
   Sys.getenv("VETIVER_API"))
60 adverse <-pin_read(board, "gt/adverse")
61
62 ```
63
```

This is important because now we have separated the often time-consuming data prep to a separate process outside of our artifact.

This process can also be repeated for anyone at the organization that is permitted to access to our pinned data. Below is the dashboard created by using pinned data:



Shiny Apps using Pinned Data

The above examples are also very important for Shiny apps where it is a best practice to separate the data wrangling logic from the Shiny app. You can learn more about this and other best practices in Joe Cheng's keynote talk here:

https://www.youtube.com/watch?v=Wy3TY0gOmJw

Separating the ETL logic from a Shiny app is important as Shiyn apps have a devoted R process that runs code dynamically. A costly ETL job can slow your app if left in your application code.

You can see our Shiny app that uses a pinned data set here:

https://github.com/philbowsher/RMD-Shiny-session-2020-4-21/blob/master/RStudio-Connect-session-2019-12-06/shiny-days-master/Pins/05-adverse-events-shiny.Rmd

Below is the code in the Shiny app that pulls in our pinned openfda data:

```r inputs}

#This to be replaced by pins pull

library(pins)
board <- board_register_rsconnect(auth = "envvar", account = "gt", server =
"http://ec2-18-216-69-78.us-east-2.compute.amazonaws.com/rsconnect/",
               key = Sys.getenv("VETIVER_API"))
adverseBase <-pin_read(board, "gt/adverse")

drugs <- unique(adverseBase$drug)


shiny::selectInput("sel_name", "Brand Name Drug", choices = drugs, selected = drugs[0])

adverse <- reactive({
  adverseBase %>%
    filter(drug == input$sel_name)
})

```

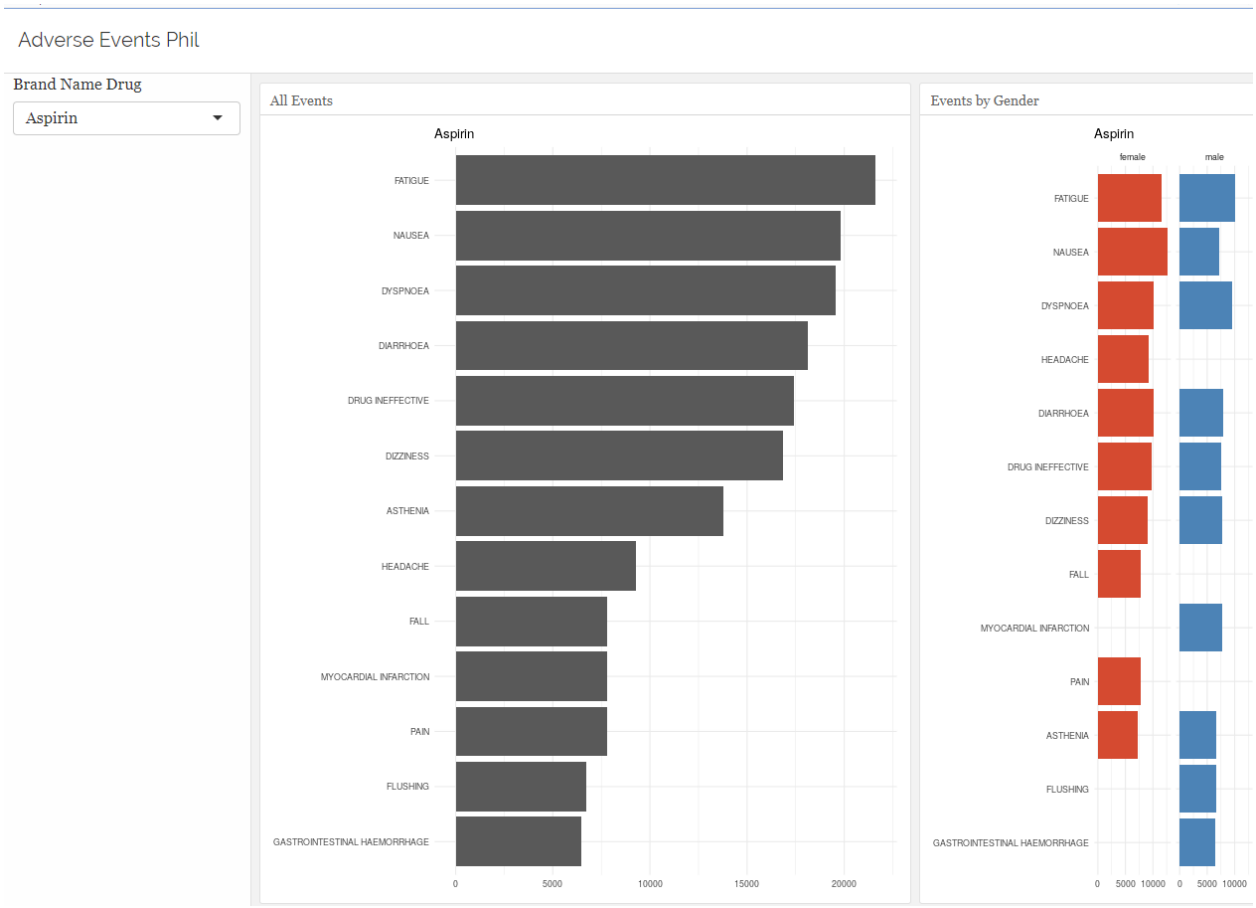
Now our pinned dataset has been converted to a Shiny reactive data frame and can be used like this:

```
### All Events

```{r}
renderPlot({
  req(adverse())
  adverse() %>%
  group_by(term) %>%
  summarise(count = sum(count)) %>%
  ggplot() +
    geom_bar(aes(reorder(term,count), count), stat = 'identity') +
    coord_flip() +
    labs(
      title = input$sel_name,
      x = NULL,
      y = NULL
    ) +
    theme_minimal()
})

```
```

The Shiny app in Production that pulls in the pinned data is below:



Contact & Summary

The information above highlights the exciting pins R package and example use cases, and how well established tools like these can help modernize clinical processes for reporting via reports, dashboards and Shiny applications.

Phil Bowsher

phil@rstudio.com

https://github.com/philbowsher