Table of contents

Welcome to Posit Team User Training!

This book is a companion to the live Posit Team User training workshop hosted by Posit.

Posit Team is a bundled offering of Posit's three professional data science tools: Posit Workbench, Posit Connect, and Posit Workbench. **For this book, we consider Posit Team users to be data scientists that are writing code and developing data products (e.g., Shiny applications and Quarto documents)**. For Posit Team administrators, we'd suggest checking out our Administrator Training.

Workshop Goal

> Workshop Goal
>
> Provide every participant with a **foundational knowledge** of Posit Team through interactive (real) data science workflows.

What is considered *foundational knowledge*? For this workshop, foundational knowledge is the *minimum* Posit Team feature set and/or workflows that every user should be aware of, including:

Posit Workbench

- Integrated Development Environment (IDE) Options
- Data Access
- Job Launcher

Posit Connect

- Publishing Methods
- Content Sharing and Access Control
- Job Scheduling
- Supported Content Types
- Runtime Settings

Posit Package Manager

- R/Python Package Access

Agenda

This workshop will loosely follow the model of a typical data science project as described in R for Data Science:

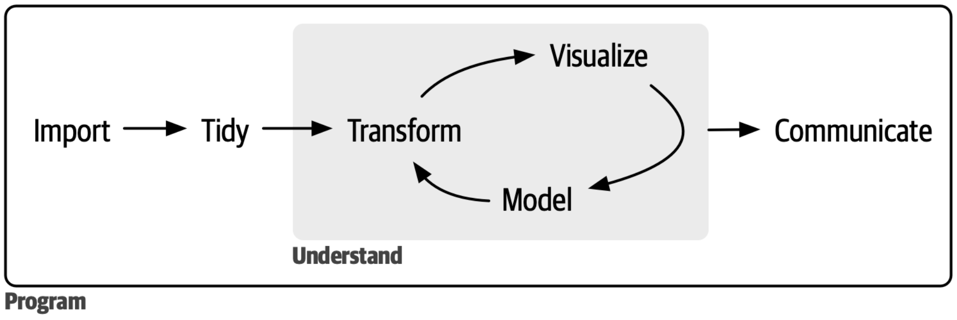


Figure 1: Data science project model

We'll begin with some workshop logistics and an overview of Posit Team. The remainder of the workshop will follow the above *Data* (focus on *Import*), *Understand*, and *Communicate* data science model. A full agenda is listed below:

- Overview and Setup
  - Accessing the workshop environment
  - Posit Team overview
  - User configuration
- Data
  - Methods for reading data into Posit Team
  - Posit Professional ODBC Drivers
  - **Exercise:** Extract-Transform-Load (ETL) workflow
- Understand
  - Creating a simple model
  - Saving and serving a model

    - **Exercise:** Create, save, and serve a model using `pins`, `vetiver`, and FastAPI

- Communicate
    - Shiny
    - **Exercise:** Publish a Shiny for Python application to Posit Connect
    - **Exercise:** Publish a Shiny for R application to Posit Connect using Git-backed deployment to Posit Connect
- Bonus Content
    - Connect API
    - `connectwidgets`
    - Job Launcher in Posit Workbench

Part I

Overview and Setup

1 Workshop Access

2 Posit Team Overview

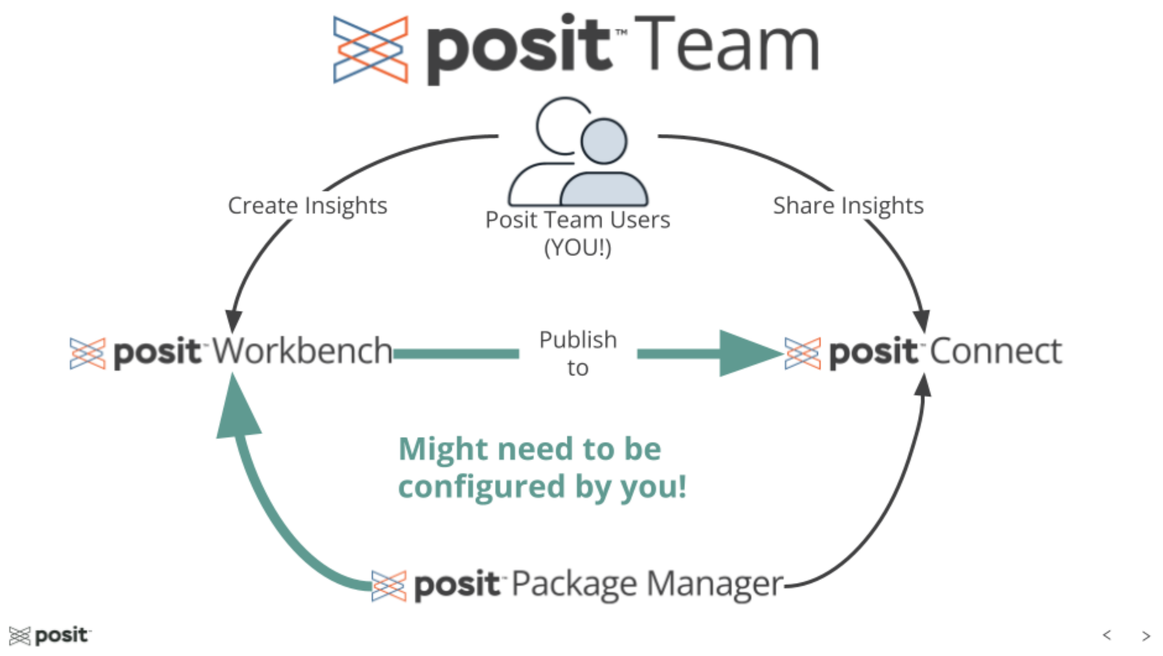


Figure 2.1: Posit Team Overview

2.1 Posit Team

Posit Team is a bundled offering of Posit's three professional data science tools including Posit Workbench, Posit Connect, and Posit Package Manager. These tools are designed to fully support end-to-end data science workflows in both R and Python from code development, sharing data products, and managing environments.

A quick overview of Posit Team and which development tools, supported content for hosting, and repository options are shown in Figure 2.2.

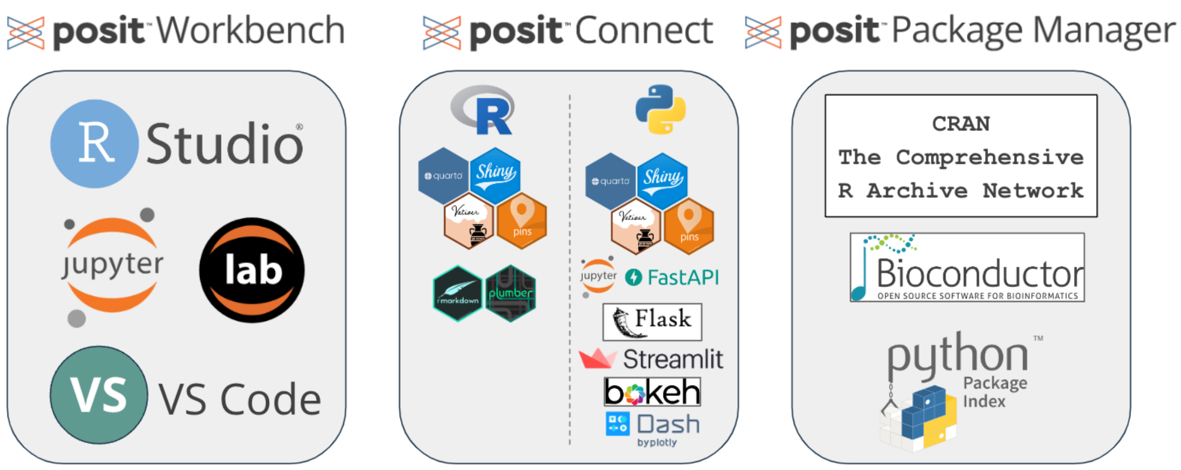
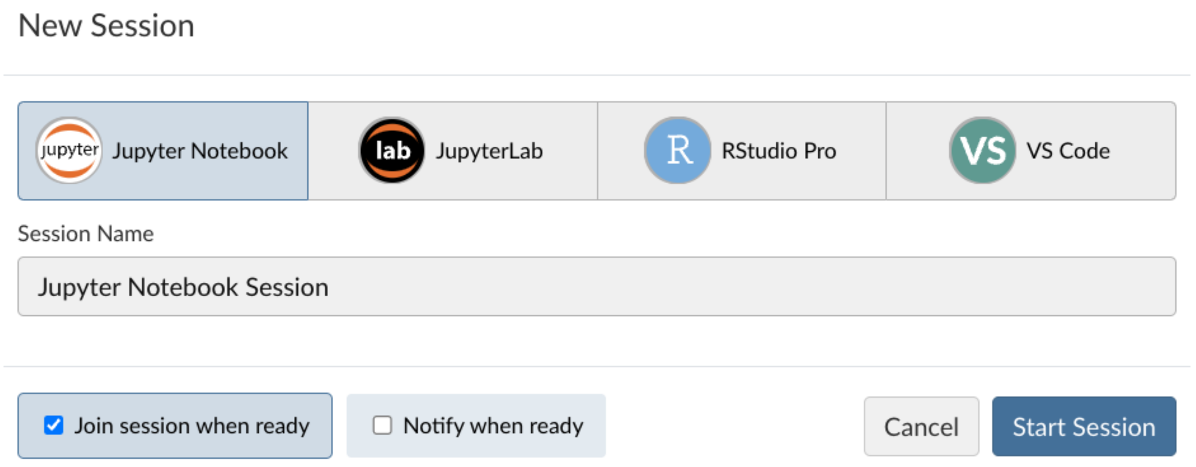


Figure 2.2: Posit Team Features

2.1.1 Posit Workbench

Data scientist leverage Posit Workbench to *create insights* (Figure 2.1). As the name implies, Posit Workbench contains numerous "tools" for insight development, including a variety of integrated development environments (IDEs) to choose from. Currently, Posit Workbench supports Jupyter Notebook, Jupyter Lab, RStudio Pro and Visual Studio Code (Figure 2.3).



New Session

Jupyter Notebook | JupyterLab | RStudio Pro | VS Code

Session Name

Jupyter Notebook Session

[✓] Join session when ready
[ ] Notify when ready

Cancel
Start Session

Figure 2.3: Posit Workbench - New session options

The IDEs included within Posit Workbench will *look* and *feel* very similar to the versions that are freely available to the public. This is by design to ensure users feel at home within Posit Workbench on day one. There are, however, a few features that you will only find within Posit Workbench which will be discussed in the next section.

2.1.1.1 Features unique to Posit Workbench

Many of the features that make Posit Workbench appealing for data science teams are tailored to system administrators, IT, and security teams including load balancing, cluster integration, authentication, session management, and security. From the users perspective, there are a handful of features that are worth knowing about which you'll only find within Posit Workbench including:

- Flexible use of multiple versions of R and Python on the same server.
- Project sharing (RStudio Pro) for easy collaboration within work groups.
- Launch long-running scripts (R and Python) in new sessions independent of your current working session.

2.1.2 Posit Connect

Data scientists leverage Posit Connect to *share insights* with others (Figure 2.1). These insights can take many forms including interactive web applications, static reports, application programming interfaces (APIs), and more. Once the insights have been published to Connect (possibly from Posit Workbench), the publisher has full control over sharing, access, and content execution.

2.1.2.1 Publishing Options

Users have multiple options for publishing content to Posit Connect, summarized in Figure 2.4.

- **Command Line:** The `rsconnect-python` package contains a command line interface (CLI) tools for publishing content from the command line/terminal. During the publishing process, the user's environment with automatically be captured and replicated on the Posit Connect server. This method is available anywhere you have access to a terminal/command line, including Jupyter Lab and VS Code.
- **Push-Button Deployment**: Available within RStudio and Jupyter Notebooks (within Posit Workbench). During the publishing process, the user's environment with automatically be captured and replicated on the Posit Connect server.
- **R:** The `rsconnect` R package makes it easy to publish content to Posit Connect from R.
- **Git**: Deploy content directly from a git repository. In this method, the user needs to supply a companion `manifest.json` file which includes information about the environment needed to run the content on Posit Connect.

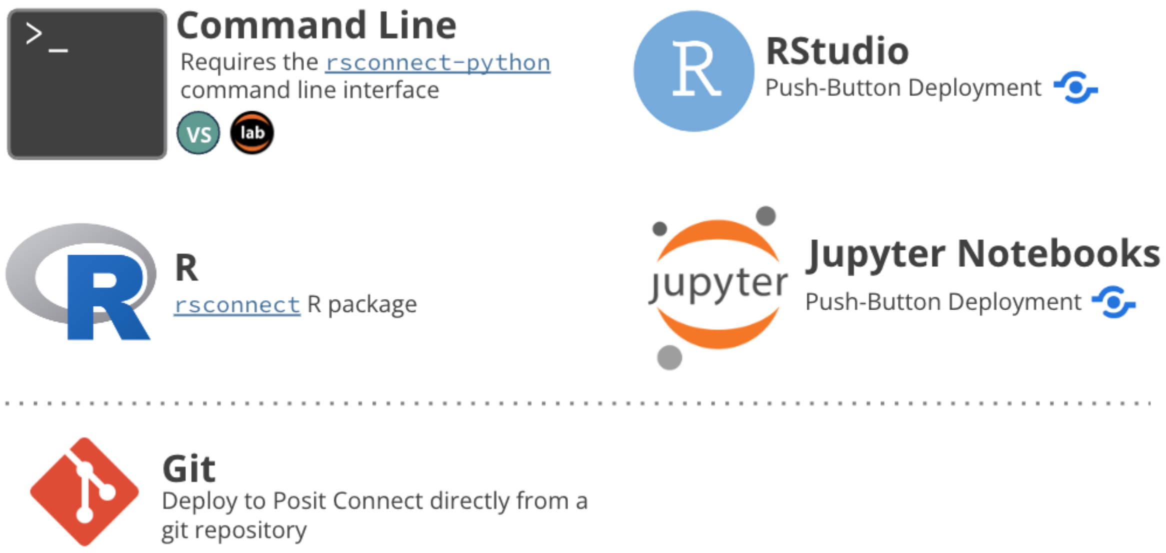


Figure 2.4: Publishing methods

2.1.3 Posit Package Manager

Data scientists leverage Posit Package Manager to access R and Python packages from repositories managed and hosted from within your organization. Often, Posit Package Manager is considered a “behind the scenes” tool, but it’s importance should not be underestimated. Users of Posit Package Manager often find that their package installs and development environments “just work” without any prior configuration. However, it’s still important that users know *how* to install packages from an instance of Posit Package Manager.

3 User Configuration

Depending on how your team has installed and configured Posit Team, there might be no user configurations needed and you can start developing, installing packages, and publishing immediately!

However, every user should be aware of which configurations are needed to make Posit Team run like well oiled machine! As show in figure Figure 2.1, users should know how to configure Posit Workbench to use packages from Posit Package Manager and how to configure a Posit Connect instance for publishing.

3.0.1 Configuring your R environment to use Posit Package Manager

Before making any configurations, you should first check which repositories you're currently using. Within an active R session, you can use the `options` function to explore which repositories ("repos") your currently using.

```
options("repos")
$repos
                                             CRAN
"https://packagemanager.posit.co/cran/latest"
```

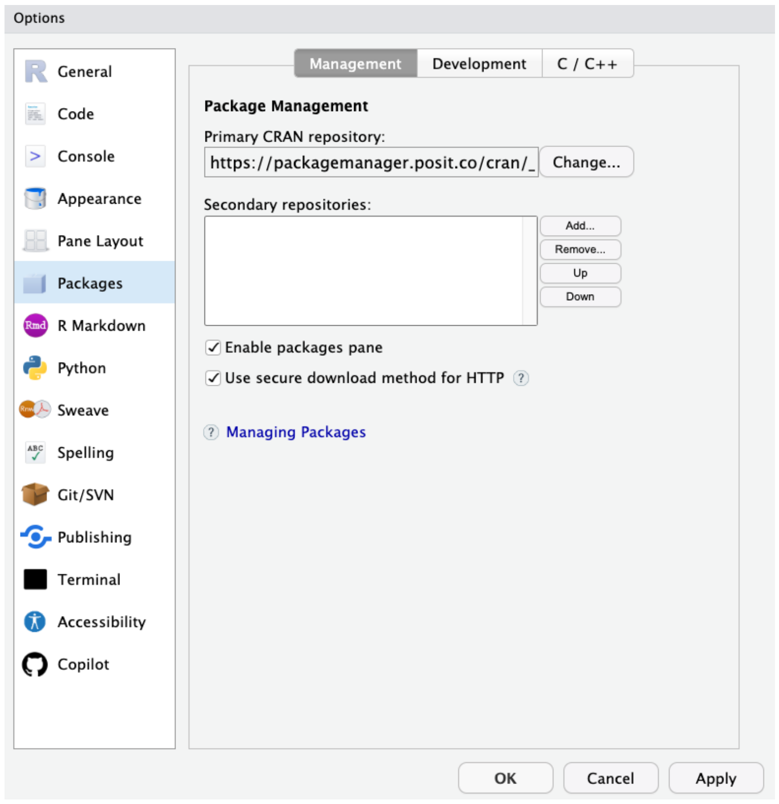
In this example, you can see the output is showing Posit's Public Package Manager as the default repository for R packages. Specifically, we are pointing to a CRAN repository hosted within package manager.

To configure R to use Posit Package Manager as it's CRAN repository, set the `repos` option to use the repository URL. The repository URL can be found within the **SETUP** page within Posit Package Manager.

```
options(repos = c(CRAN = "https://packagemanager.posit.co/cran/__linux__/centos7/latest"))
```

Since this configuration is done within an active R session, you'll need to repeat these steps anytime you open a new session. Consider adding this code to your `.Rprofile` file to maintain the repository configuration across R sessions.

You can also set the repository URL using the RStudio IDE by navigating to **Tools –> Global Options –> Packages**, and adding the repository URL to the "Primary CRAN repository" text box.



Package Management

Primary CRAN repository:
https://packagemanager.posit.co/cran/_

Secondary repositories:

✓ Enable packages pane

✓ Use secure download method for HTTP

Figure 3.1: RStudio Global Options - Packages

3.0.2 Configuring your Python environment to use Posit Package Manager

Pip is commonly used to install and manage the Python packages in your environment. To see which repositories pip is currently pointing to, we can run `pip config list` from the command line/terminal:

```
pip config list
global.index-url='https://packagemanager.posit.co/pypi/latest/simple'
```

The `global.index-url` let's us know which repository pip will used for python package installation. In this case, it's a mirror of the Python Package Index (PyPI) within Posit Package Manager.

You can modify the `global.index-url` setting for all python projects by running:

```
pip config set global.index-url https://packagemanager.posit.co/pypi/latest/simple
```

3.1 Configure a Posit Connect instance for publishing

Publishing your data products to Posit Connect is arguably the most valuable feature of Posit Team. Configuring a Posit Connect instance will vary depending on your development environment and publishing method (see Figure 2.4). For this workshop, we will cover push-button deployment via the RStudio IDE and CLI publishing. Additional details for configuring a Posit Connect instance can be found on the Posit Connect User Guide.

3.1.1 Push Button Deployment (RStudio)

Navigate to **Tools –> Global Options –> Publishing.** In some cases, the Posit Connect will already be configured for you. To add a new Posit Connect instance, click the **Connect** button and follow the instructions. The only thing needed is the Posit Connect URL which can be supplied by your System Administrator.

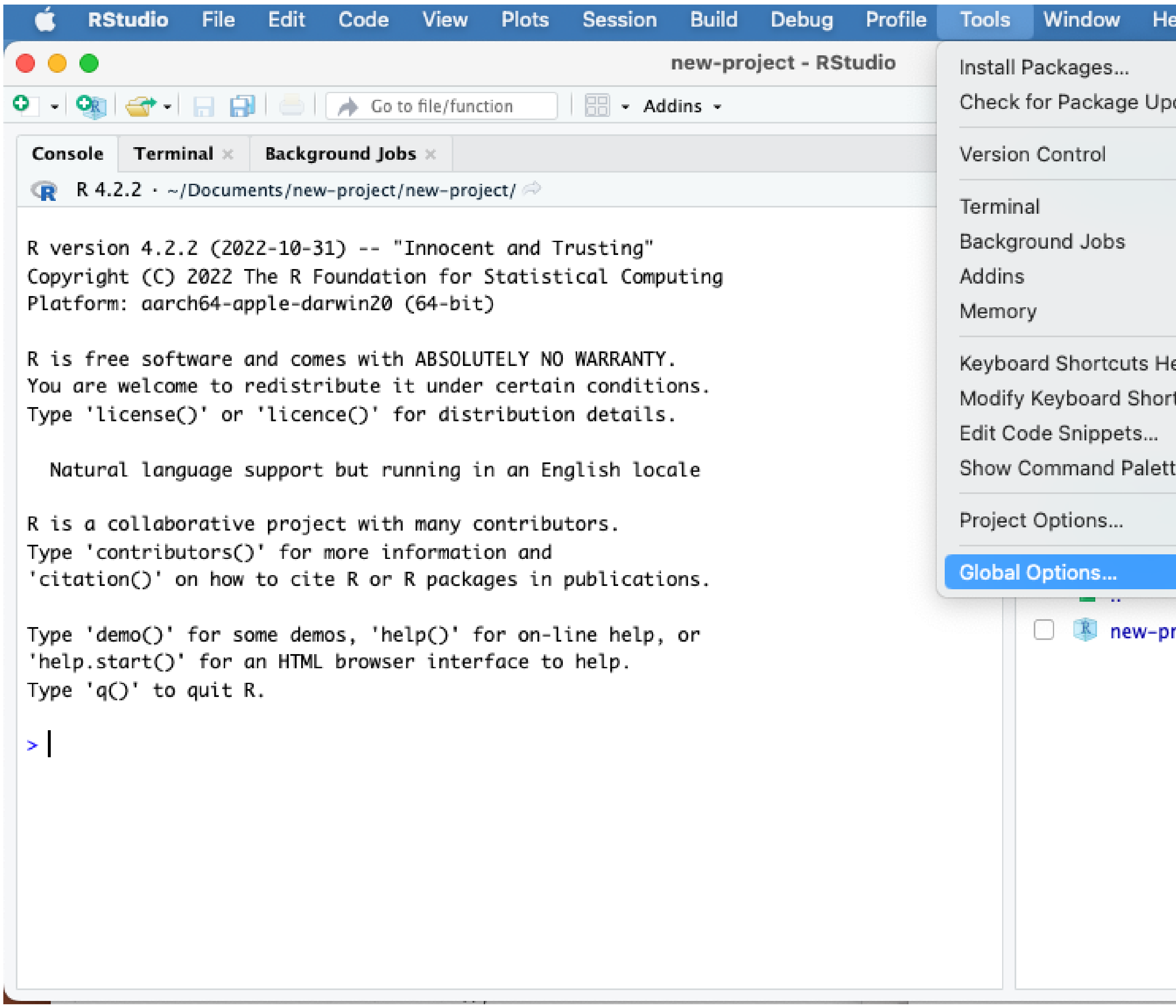
3.2 CLI Publishing

To check currenctly connected Posit Connect servers, you can use the `rsconnect-python` CLI tool and run the following command from the terminal/command line:

```
rsconnect add list
```

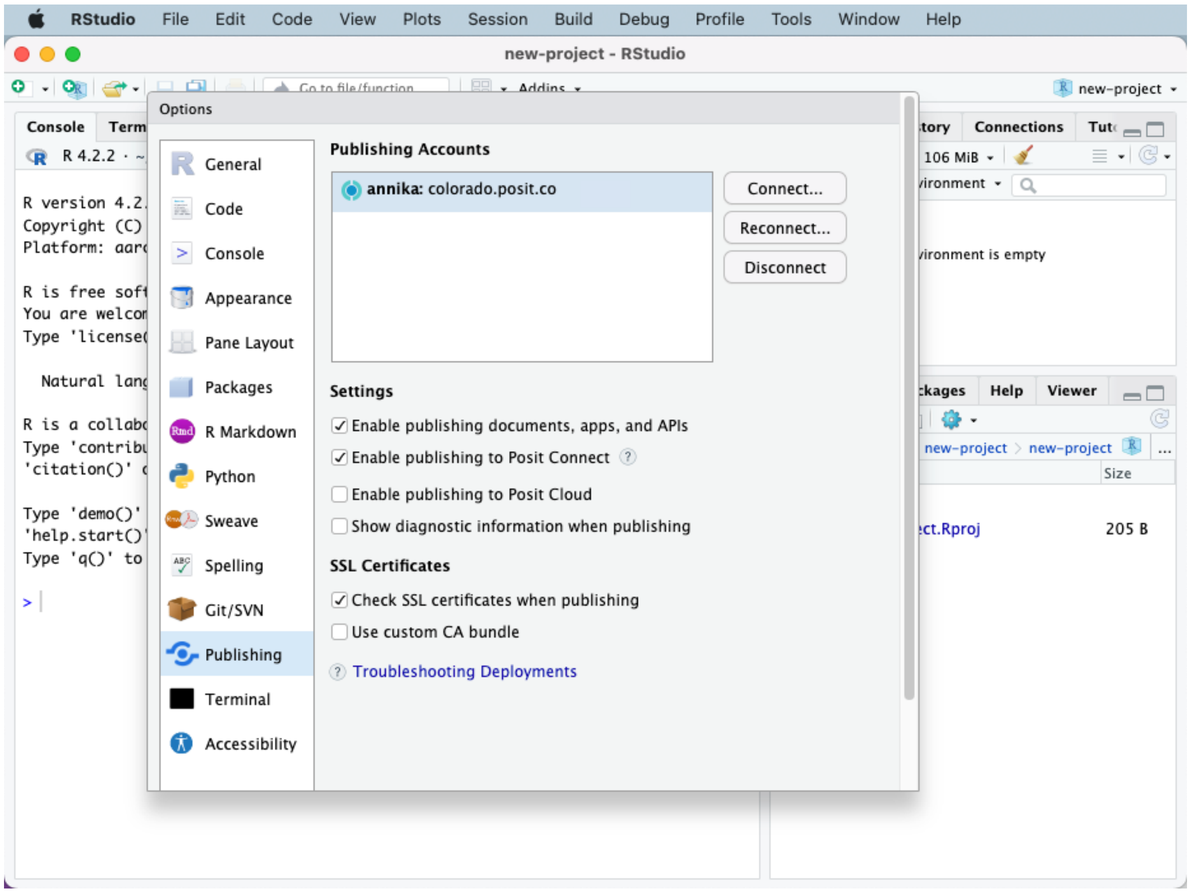
To add a new Posit Connect instance, you'll need the Posit Connect server URL as well as an API key (instructions for adding an API can be found here). You also need to give your Posit Connect instance a name.

```
rsconnect add \
    --server https://my.connect.server/ \
    --name myServer \
    --api-key myconnectapikey
```



```
R version 4.2.2 (2022-10-31) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Figure 3.2: RStudio - Configure Posit Connect Instance - Step1



Publishing Accounts

annika: colorado.posit.co

Settings

- [✓] Enable publishing documents, apps, and APIs
- [✓] Enable publishing to Posit Connect
- [ ] Enable publishing to Posit Cloud
- [ ] Show diagnostic information when publishing

SSL Certificates

- [✓] Check SSL certificates when publishing
- [ ] Use custom CA bundle

Troubleshooting Deployments

Figure 3.3: RStudio - Configure Posit Connect Instance - Step 2

Part II

Data

4 Data

"Data is the new oil!"

- British mathematician Clive Humbly, 2006

Data is a precious commodity, playing a key role in informing decisions and driving insights. Similar to oil, if data is not properly mined, it's essentially useless. In order for the value of data to be realized, data science teams need a way to store and access data freely and readily.

4.1 Data Import Options

Data lives in a variety of locations in a variety of formats, and as such, there are many options for bringing your data into your development environment. Below are few common options for accessing your data:

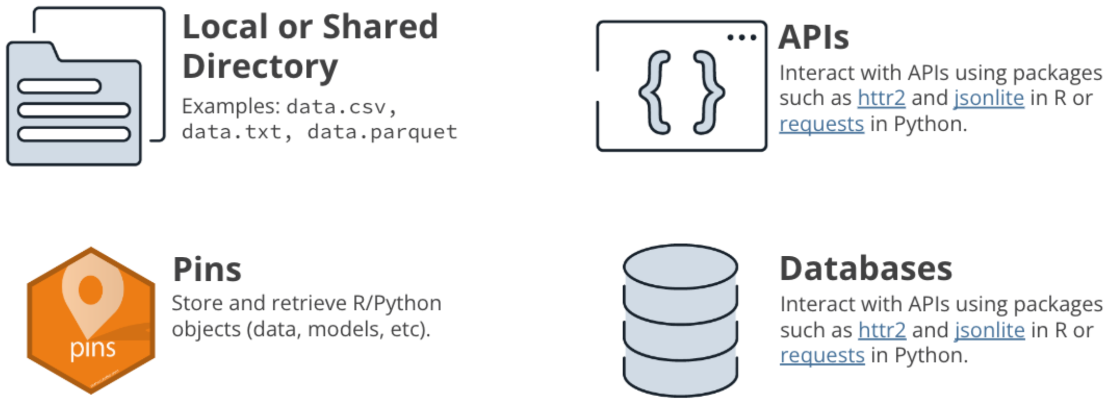

Local or Shared Directory
Examples: data.csv, data.txt, data.parquet
APIs
Interact with APIs using packages such as httr2 and jsonlite in R or requests in Python.
pins
Pins
Store and retrieve R/Python objects (data, models, etc).
Databases
Interact with APIs using packages such as httr2 and jsonlite in R or requests in Python.

Figure 4.1: Data import options

Of the four options listed in Figure 4.1, Databases are the most complex. In the next section, we will give you a primer on working with databases.

4.2 A Primer on Databases

Databases are managed by **D**ata**B**ase **M**anagement **S**ystems (DBMSs). There exists a growing list of DBMSs for databases hosted on-prem (client server), in the Cloud, or in-process. Below are a few common DBMSs:

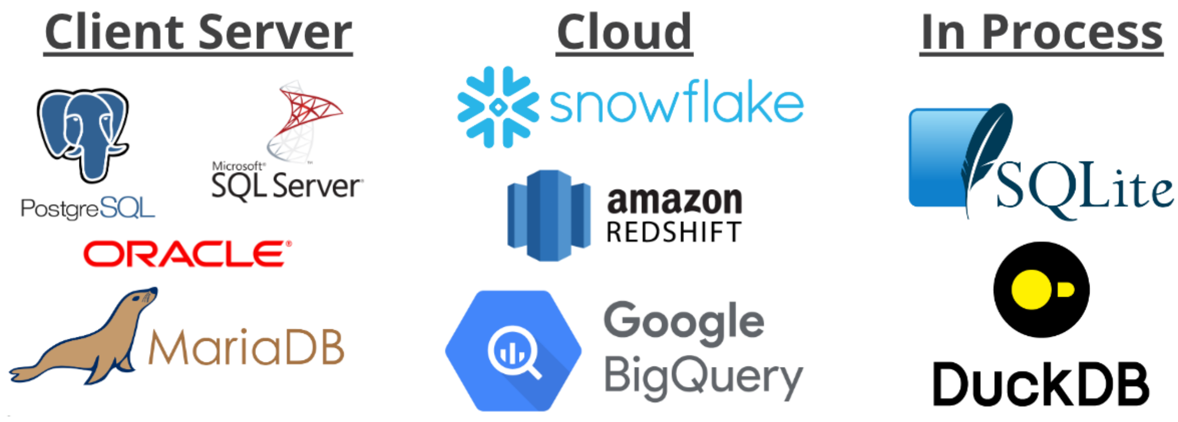


Figure 4.2: Database Management Systems

In R, there exists an *interface* layer between the R programming language and the DBMS. This is know as the **D**ata**B**ase **I**nterface (DBI). To make a connection to a DBMS from R, you use the `DBI::dbConnect()` function in combination with an R package tailored for the DBMS you are connecting to. For example, here is some R code to connect to a PostgreSQL database:

```r
con <- DBI::dbConnect(
  RPostgres::Postgres(),
  hostname = "databases.mycompany.com",
  port = 1234
)
```

In Python, the database interface layer is an API known as the Python **D**ata**B**ase **API** (DB-API). Most packages and modules in Python used to access DBMSs are often designed to be DB-API compliant including `sqlite3`, `psycopg2`, and `mysql-connector-python`. For example, here is some Python code to connect to a SQLite database:

```python
con = sqlite3.connect('example.db')
```

With so many DBMS-specific connectors, it can be overwhelming to remember which package/function to use. As such, most DBMSs will also provide a connector known as an **O**pen **D**ata**B**ase **C**onnectivity (ODBC) driver. ODBC is a *universal* DBMS interface, which means the same ODBC functions will work with any database. For example, here is how to make an ODBC connection to a PostgreSQL database in R and Python:

```
# In R
library(DBI)
library(odbc)

con <- DBI::dbConnect(odbc::odbc(),
  driver = "PostgreSQL Driver",
  database = "test_db",
  UID    = Sys.getenv("DB_USER"),
  PWD    = Sys.getenv("DB_PASSWORD"),
  host = "localhost",
  port = 5432)
```

```
# In Python
import pyodbc

con = pyodbc.connect(
  driver = 'PostgreSQL',
  database = 'test_db',
  server = 'localhost',
  port = 5432,
  uid = os.getenv('DB_USER'),
  pwd = os.getenv('DB_PASSWORD')
)
```

4.3 Posit's Professional ODBC Drivers

Not all DBMSs will provide an ODBC Driver. This can be a major blocker for users attempting to import data into their development environment. To overcome this limitation, Posit has created custom ODBC drivers that work with many of the popular DBMSs used today, including:

Make sure to speak with your Posit Team system administrator if you currently use any of the above databases.

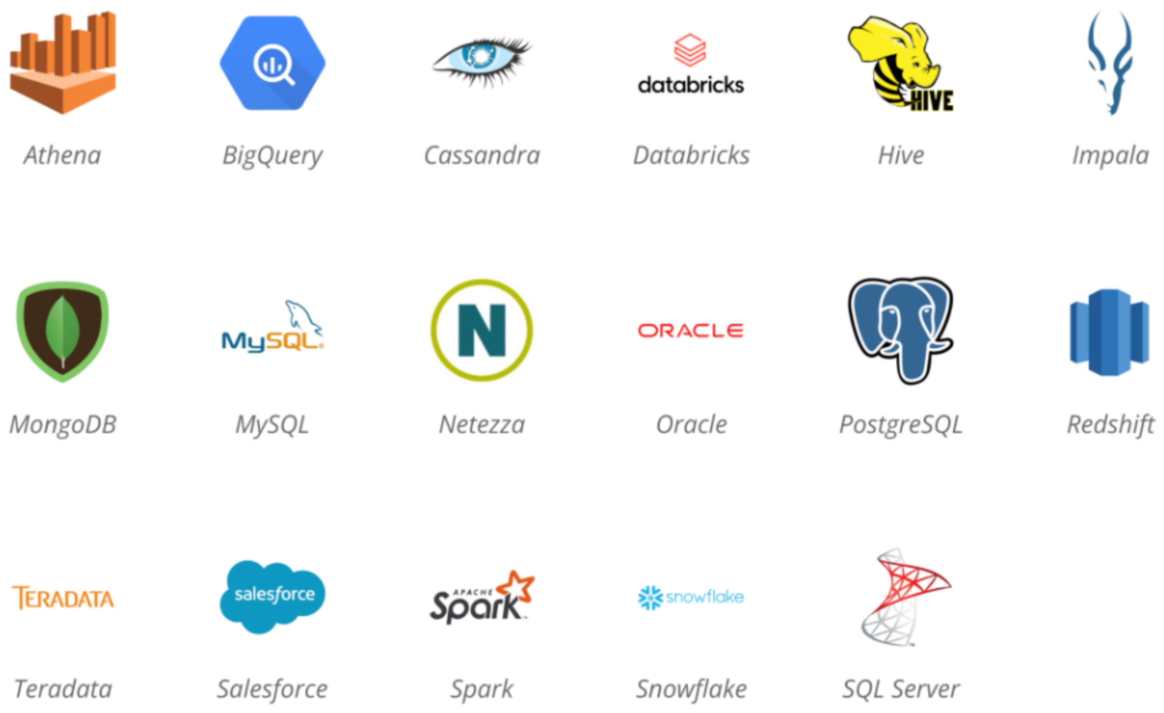


Figure 4.3: Posit's Professional ODBC Drivers

5 Exercise: ETL

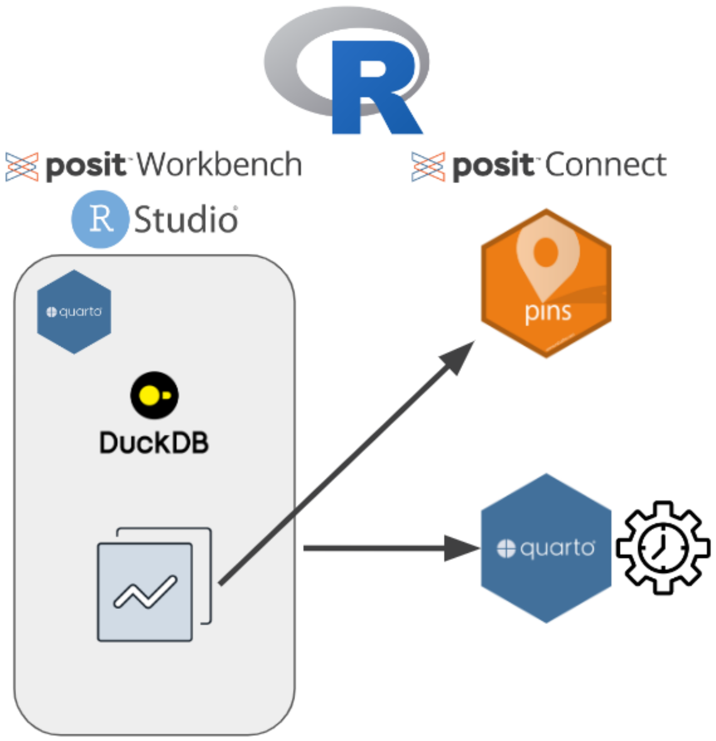


Figure 5.1: ETL Workflow

A common data science workflow entails **E**xtracting data, **T**ransforming it, and then **L**oading (ETL) it to a location to be shared with others or consumed by other content. In this exercise, you will get practice creating an ETL workflow within Posit Team. You will also leverage the job scheduling feature in Posit Connect to ensure the data remains updated automatically.

The data for this exercise is "real" data regarding cases of COVID19 in the United States from 2020 to 2023. While this is a relatively small dataset, the ETL workflow (as shown in Figure 5.1) is designed to work with data of all sizes! Here is a preview of the COVID dataset with an explanation of the various columns:

```
# A tibble: 66,100 x 4
```

```
   province_state date       state_count new_cases
   <chr>          <date>           <dbl>     <dbl>
 1 Alabama        2020-01-23           0         0
 2 Alabama        2020-01-24           0         0
 3 Alabama        2020-01-25           0         0
 4 Alabama        2020-01-26           0         0
 5 Alabama        2020-01-27           0         0
 6 Alabama        2020-01-28           0         0
 7 Alabama        2020-01-29           0         0
 8 Alabama        2020-01-30           0         0
 9 Alabama        2020-01-31           0         0
10 Alabama        2020-02-01           0         0
# i 66,090 more rows
```

`province_state`: State or province in the United States of America.

`date`: Date of reporting.

`state_count`: Total number of confirmed COVID cases in state as of `Date`.

`new_cases`: Number of new confirmed COVID cases on `Date`.

5.1 Step 1 - Extract Data from DuckDB

The workshop environment comes with a DuckDB, which can be be found here: `duckdb/database/demo-datasets.db`. Within the DuckDB database is the COVID dataset described in . The first step is to make a connection to the DuckDB and read in the COVID data using R.

All of this R code should be added to a Quarto document within the RStudio IDE on Posit Workbench.

5.1.1 Load Necessary Packages

```r
library(DBI)
library(dplyr)
library(pins)
library(duckdb)
library(dbplyr)
```

5.1.2 Connect to DuckDB

For this connection, we are using the `DBI` package in combination with the `duckDB` package.

```r
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = "/data/duckdb/database/demo-datasets.db")
```

After the connection (`con`) has been made, you can explore the various dataset contained within the DuckDB by running:

```r
dbListTables(con)
```

5.1.3 Extract/Transform Covid Data

Let's use the `dplyr::tbl()` function to extract the Covid data from the DuckDB connection. The `tbl()` makes a "pointer" to the COVID data within the DuckDB. To pull the data into our active R session and store it in memory, we need to use the `dplyr::collect()` function:

```r
covid <- tbl(con, "covid") |>
  # Read into memory
  collect()
```

In order to save compute resources in your R session, it's suggested to do any type of cleaning and filtering of the data before calling the `collect()` function. Therefore, let's amend the previous code and insert a *transformation* step before the `collect()` function to filter our data for a specific state/province. Feel free to substitute "Maryland" in the code below for another state/province:

```r
covid <- tbl(con, "covid") |>
  # Filter for a state/provice
  filter(province_state == "Maryland") |>
  # Read into memory
  collect()
```

5.2 Step 2 - Load Data to Posit Connect

Now that the data has been transformed (filtered for a specific state/province), it's time to *load* the data so that other users/content can access it. For this exercise, we are going to use the popular `pins` package to load the data to Posit Connect.

Pinning an object to Posit Connect is a two step process. The first step is to define Posit Connect as our pinning *board*:

```
board <- pins::board_connect()
```

For this workshop, the `board_connect()` function should work without supplying any arguments. When running this command in your own environment, you may need to do some configuration which is described here.

The second step is to write the data to Posit Connect as a pin. Before running the code below, take note of what your username is on the Posit Connect server. In this example, I'm using the placeholder name of `publisher1`, but be sure to replace this with **your** username!

```
pins::pin_write(board = board, x = covid, name = "publisher1/covid_data", type = "csv")
```

5.3 Step 3 - Publish Workflow and Automate

Although the COVID dataset is static, from early 2020 to mid 2023, it was being updated daily. If the data changes, you'll need to re-run the ETL workflow described above to ensure the pinned dataset reflects the latest data. This is a manual process that we can automate with the help of Posit Team!

First, click the blue deployment button at the top of the RStudio IDE:

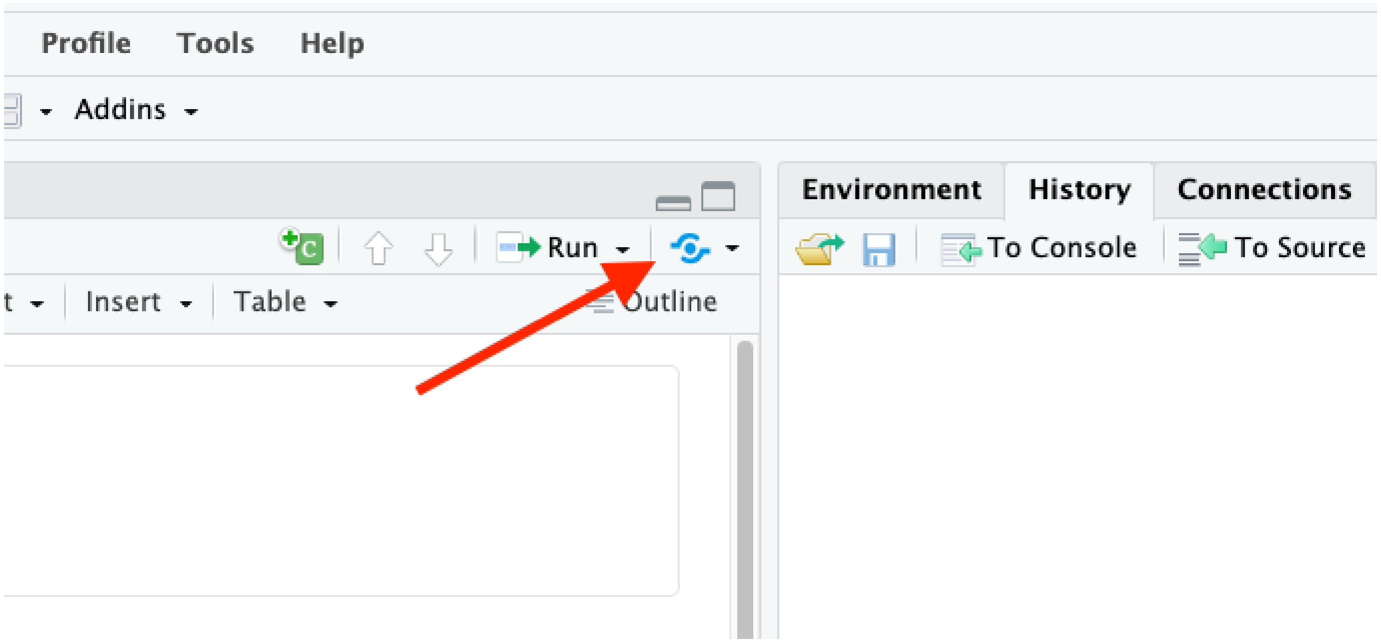
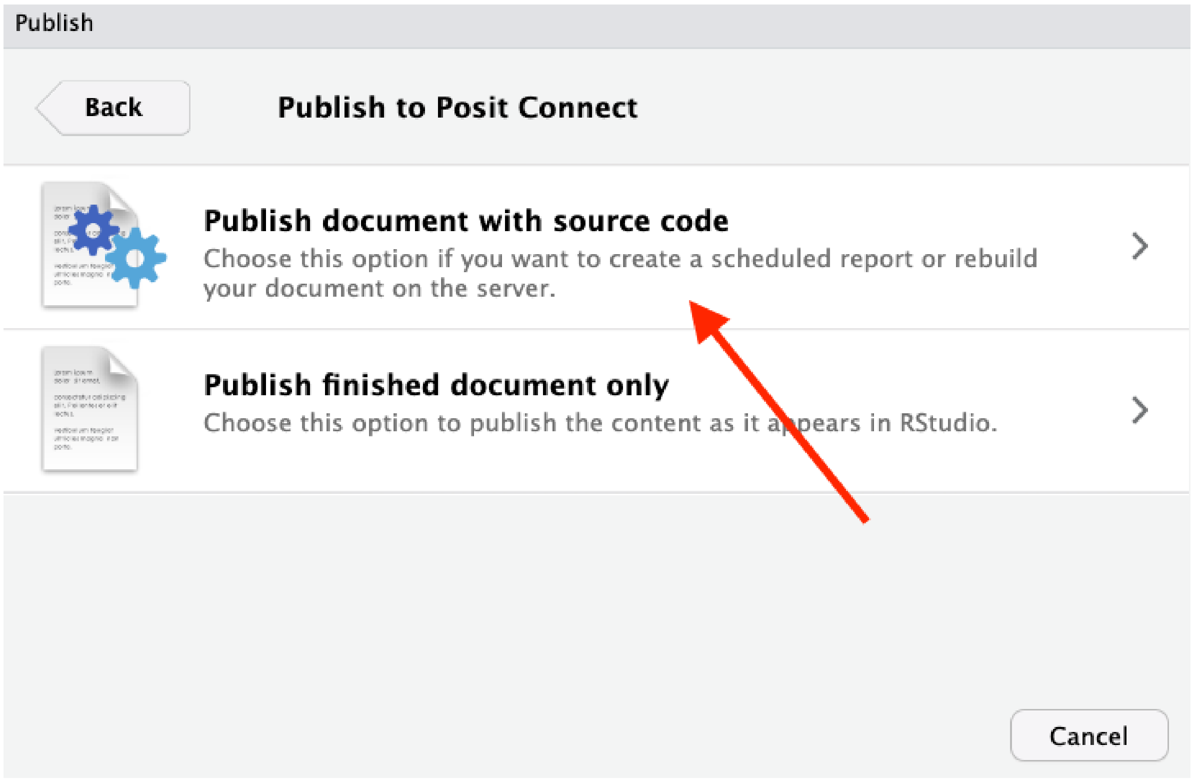


Figure 5.2: RStudio - Deployment Button

In the subsequent pop-up window, select **Posit Connect.** Importantly, to re-run the quarto document on the Connect server, we need to **publish document with source code**:



Publish document with source code
Choose this option if you want to create a scheduled report or rebuild your document on the server.

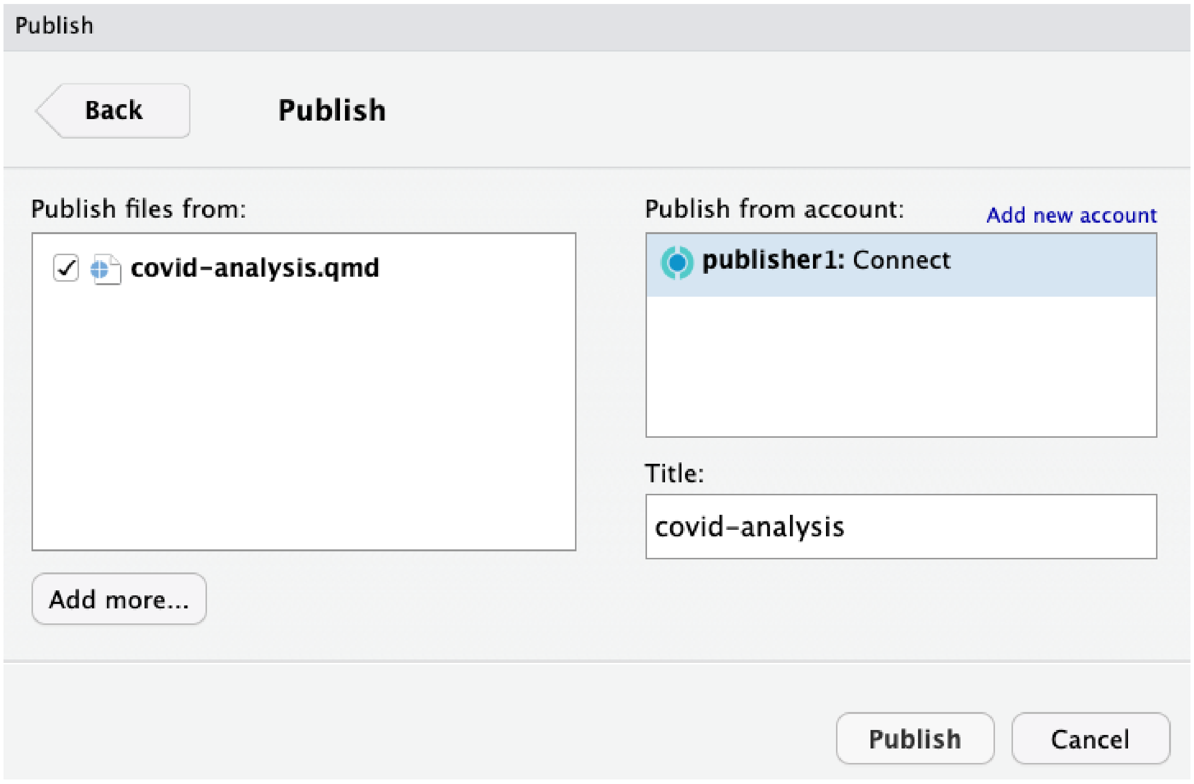
Publish finished document only
Choose this option to publish the content as it appears in RStudio.

Figure 5.3: RStudio - Publish Options

The last step is to define the files you want to send to Connect, the Connect instance you want to publish to, and the title of the content. If everything looks good, go ahead and click **Publish**!

Once hosted on Connect, let's set the Quarto document to re-run on a daily basis. We accomplish this by selecting the *Schedule* tab at the top. In the below configuration, the Quarto document will run every day at 9:48 am (America - New York time) and will ensure the pinned dataset is always up-to-date!

You'll also notice at the bottom we have two options to publish the output after it is generated and send email after update. The first option ensure that the content on Posit Connect is updated after it's scheduled to run. This is important if your content has any plots or tables that need to be updated. The second option allows you to send an email to users with a direct link to the content on Posit Connect. This is a great option to deliver insights for those that love to live in their email inbox .



Publish files from:

- [✓] covid-analysis.qmd

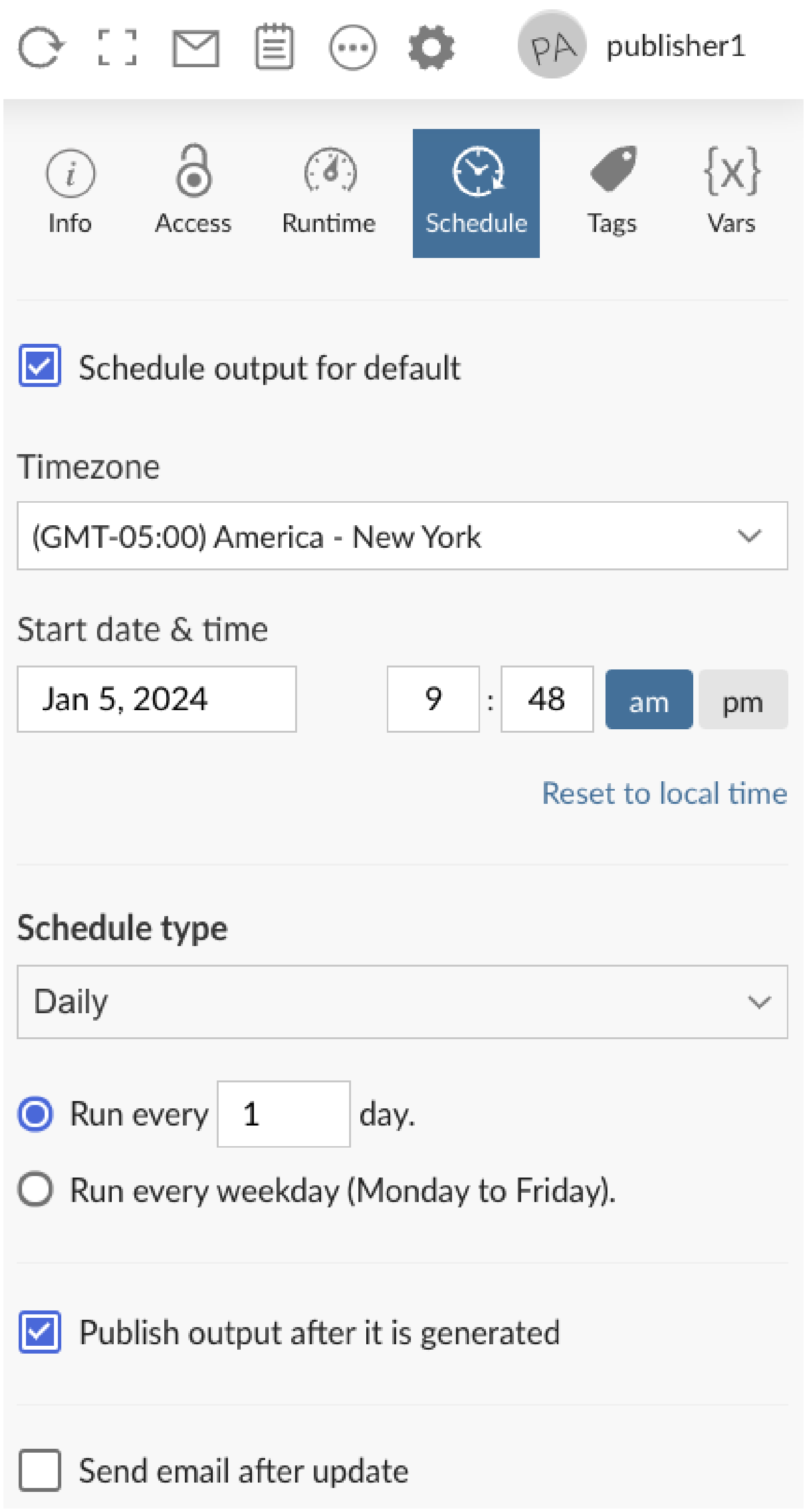
Publish from account:

publisher1: Connect

Title:

covid-analysis

Figure 5.4: RStudio - Publish to Posit Connect



[✓] Schedule output for default

Timezone
(GMT-05:00) America - New York

Start date & time
Jan 5, 2024 9 : 48 [✓ am] pm

Reset to local time

Schedule type
Daily

[✓ Run every 1 day.]
Run every weekday (Monday to Friday).

[✓] Publish output after it is generated

[ ] Send email after update

Figure 5.5: Job Scheduling on Posit Connect

Part III

Understand

6 Modeling with Posit Team

Many data science workflows culminate in a deliverable. This could be a single plot or table, a detailed dashboard, or a interactive web application. Deliverables must tell a story, and the only way to tell a good story is to **understand** the underlying data.

Machine Learning (ML) is an exciting and ever-evolving facet of data science, offering a valuable means to extract insights from your dataset that may not be readily apparent through basic plots and tables. While generating models can pose certain challenges, the greater challenge often lies in determining the methods to save, distribute, and serve your model so that others can interact with it. In this chapter, we'll discuss how Posit Team can be used to support the entire ML life cycle, from creation to delivery!

6.1 The Machine Learning Life Cycle

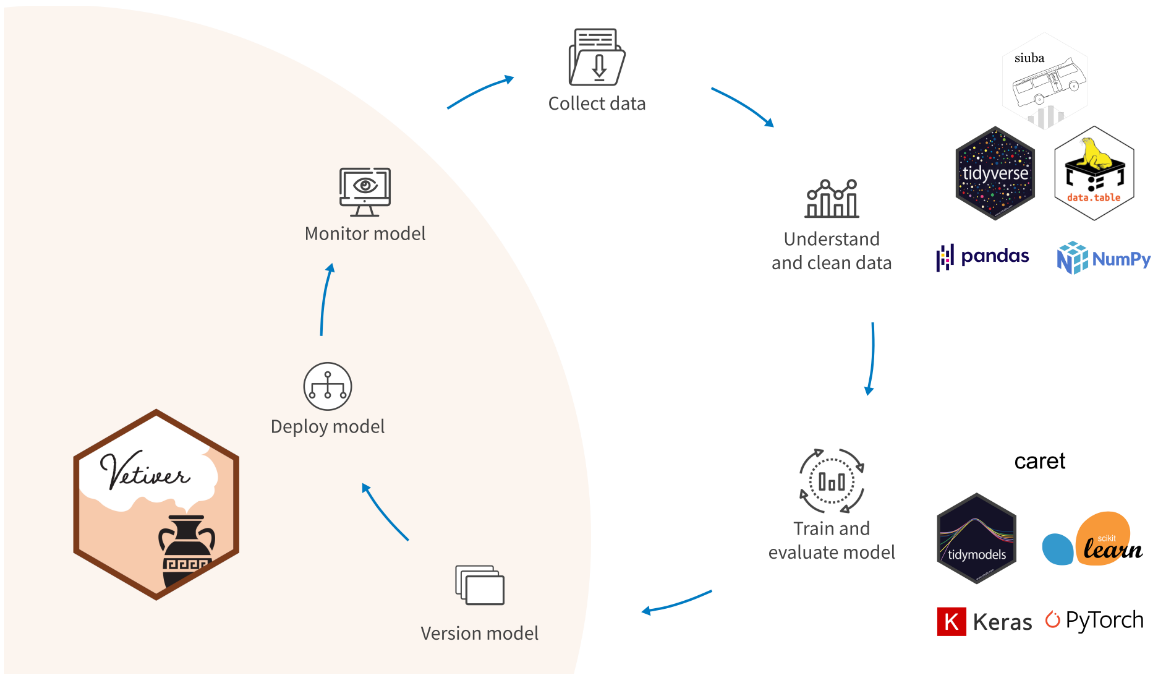


Figure 6.1: Machine Learning Life Cycle

As seen in Figure 6.1, all models start with data (very top of the cycle). The next step is to prepare the data for modeling, which requires cleaning and transforming to ensure the data is in a format that is usable for machine learning. Which tool(s) you use to create an ML model is completely up to you, but here's a short list of common R/Python tools:

Table 6.1: Packages, Libraries, and Frameworks for ML Model Creation

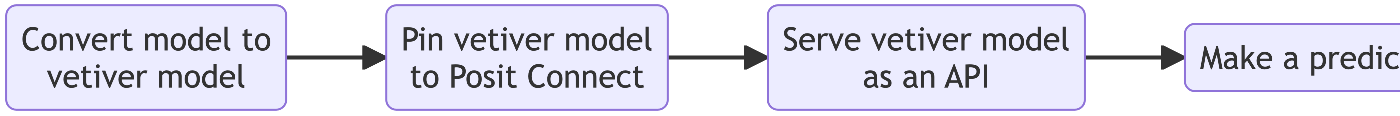
R	Python
`tidymodels`	SciKit-Learn
`caret`	Keras
	PyTorch
	TensorFlow

Rarely is the creation of a model a one time event. As data changes and the ML tools evolve, models will need to be updated and subsequently deployed into production. It's also important that the deployed model demonstrates high performance compared to prior versions and other models.

6.2 MLOps with `vetiver`

Given the iterative nature of the ML life cycle, there exists tools explicitly designed to assist with the deployment and maintenance (i.e. "operations") of ML models. This practice is known as **M**achine **L**earning **O**perations, or **MLOps**. Posit has created a MLOps framework for both R and Python known as `vetiver`. `vetiver` was designed to fill the tool gap in the ML life cycle around versioning, deploying, and monitoring model performance.

6.2.1 `vetiver` workflow with Posit Team


Convert model to vetiver model
Pin vetiver model to Posit Connect
Serve vetiver model as an API
Make a predic

A typical `vetiver` workflow within Posit Team, depicted in the flow diagram above, consists of converting a model to a *vetiver model*, saving it to Posit Connect as a `pin`, and then serving it as an API. This workflow is shown below in both Python and R. Be sure to substitute "your_name" with your Posit Connect username:

6.3 Python

```python
import pins
from vetiver import VetiverModel
from vetiver.data import mtcars
from sklearn.linear_model import LinearRegression

# Create Model
model = LinearRegression().fit(mtcars.drop(columns="mpg"), mtcars["mpg"])

# Create Vetiver Model
v = VetiverModel(model, model_name = "your_name/cars_linear",
                 prototype_data = mtcars.drop(columns="mpg"))

# Save Model as pin
board = pins.board_connect(allow_pickle_read = True)
vetiver_pin_write(board, v)
```

6.4 R

```r
library(vetiver)
library(pins)

# Create Model
cars_lm <- lm(mpg ~ ., data = mtcars)

# Create Vetiver Model
v <- vetiver_model(cars_lm, "your_name/cars_linear")

# Save Model as pin
board <- board_connect()
vetiver_pin_write(board, v)
```

Now that we have a model *saved* to Posit Connect, we can use `vetiver` to serve it as an API. By default, `vetiver` will use FastAPI for python models, and `plumber` for R models. Creating APIs is a cinch with `vetiver`:

6.5 Python

```python
from vetiver import deploy_rsconnect
from rsconnect.api import RSConnectServer
import os

# Define Connect Server
connect_server = RSConnectServer(
    url=os.getenv("CONNECT_SERVER"),
    api_key=os.getenv("CONNECT_API_KEY")
)

# Convert pinned model on Connect to API
deploy_rsconnect(board = board,
                 pin_name = "you_name/cars_linear",
                 connect_server = connect_server)
```

6.6 R

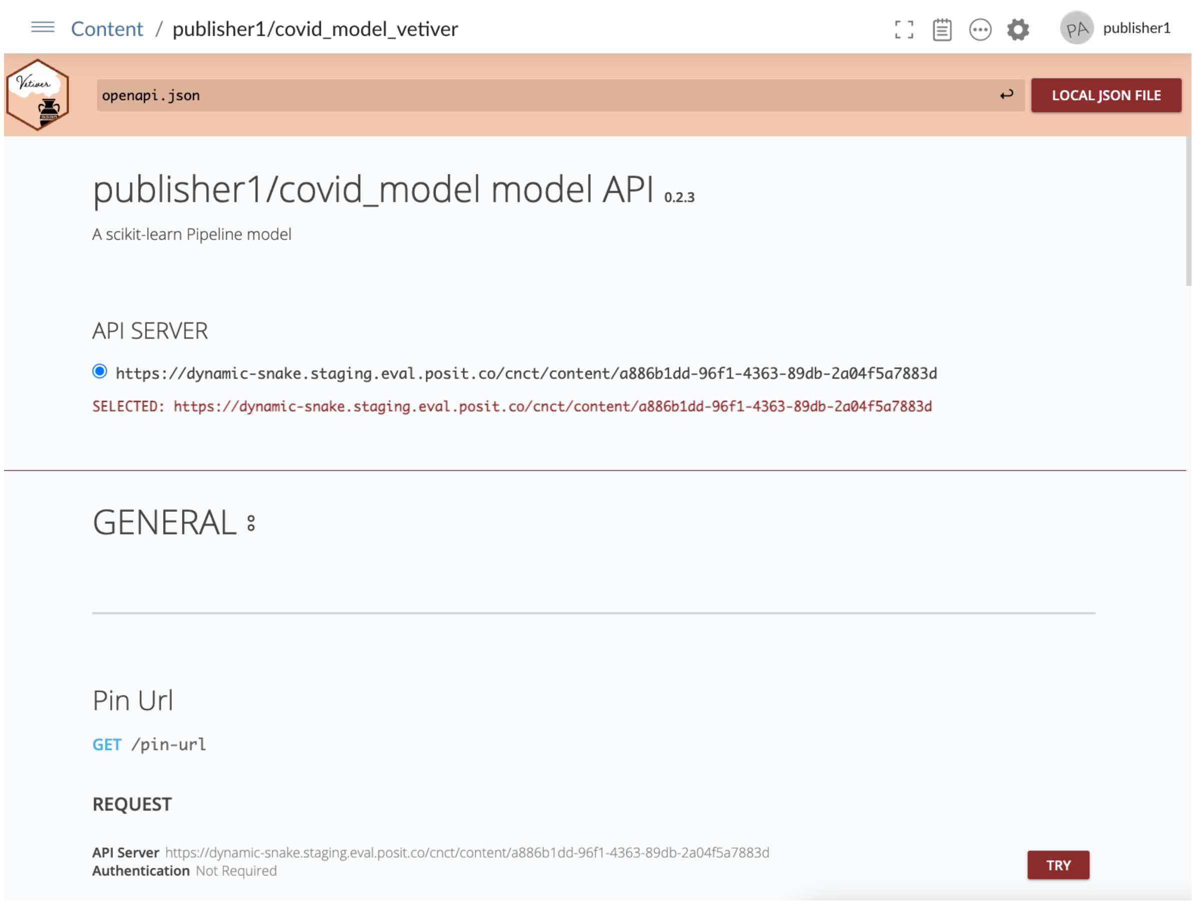
```r
library(plumber)
library(rsconnect)

# Convert pinned model on Connect to API
vetiver_deploy_rsconnect(board = board,
                         name = "your_name/cars_linear")
```

In some cases, you may need to add some additional arguments to these functions (e.g., Posit Connect server URL and API key). Additional details can be found on the Posit Connect user guide.

6.6.1 Interacting with APIs on Posit Connect

Human interactions with an API usually requires a visualization layer. When creating an API for your model with `vetiver`, the default visual documentation for Plumber is *swagger*, and for FastAPI it's *RapiDoc*. Below is an example of the RapiDoc documentatin of a FastAPI hosted on Posit Connect:

This example API uses a model to predict the number of COVID cases given a specific day of the year. You will create a similar API in the next exercise! The prediction is generated by



publisher1/covid_model model API 0.2.3

A scikit-learn Pipeline model

API SERVER

https://dynamic-snake.staging.eval.posit.co/cnct/content/a886b1dd-96f1-4363-89db-2a04f5a7883d

SELECTED: https://dynamic-snake.staging.eval.posit.co/cnct/content/a886b1dd-96f1-4363-89db-2a04f5a7883d

GENERAL

Pin Url

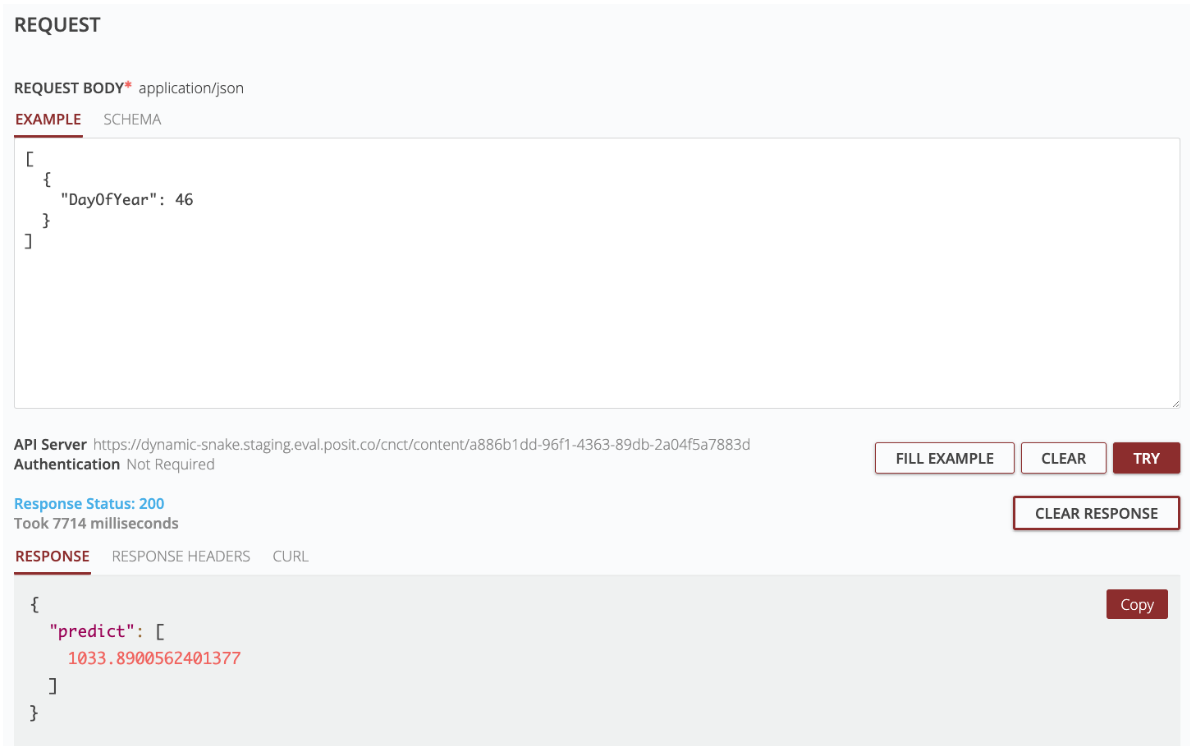
GET /pin-url

REQUEST

API Server https://dynamic-snake.staging.eval.posit.co/cnct/content/a886b1dd-96f1-4363-89db-2a04f5a7883d
Authentication Not Required

Figure 6.2: RapiDoc documentation for a FastAPI hosted on Posit Connect

sending a **POST** request to the API with specific parameters. In this case, there is only one parameter, the day of the year (`DayOfYear`). RapiDoc allows you to try out the API. In the image below, we asked the API to return the prediction of the number of COVID cases on day 46 of the year. At the bottom, you can see the response of approximately 1,033 cases:



```
[
  {
    "DayOfYear": 46
  }
]
```

API Server https://dynamic-snake.staging.eval.posit.co/cnct/content/a886b1dd-96f1-4363-89db-2a04f5a7883d
Authentication Not Required

Response Status: 200
Took 7714 milliseconds

```
{
  "predict": [
    1033.8900562401377
  ]
}
```

Figure 6.3: Interacting with RapiDoc

You can also interact with APIs programmatically. In the code below, we'll show you how to query an API using `vetiver` hosted on Posit Connect using both R and Python. Full instructions (including additional authentication options) can be found here.

6.7 Python

```python
from vetiver.server import predict, vetiver_endpoint

# Define the API endpoint
endpoint = vetiver_endpoint(
    f"https://connect.example.com/content/{APP_ID}/predict"
)
```

```
# If API has restricted access, you'll need to supply an API Key
api_key=os.getenv("CONNECT_API_KEY")

# Predict using new data!
h = {"Authorization": f"Key {api_key}"}
response = predict(endpoint=endpoint, data=new_data, headers=h)
```

6.8 R

```
# Define the API endpoint
endpoint <- vetiver_endpoint(
  "https://connect.example.com/content/$APP_ID/predict")

# If API has restricted access, you'll need to supply an API Key
apiKey <- Sys.getenv("CONNECT_API_KEY")

# Predict using new data!
predict(
  endpoint,
  new_data,
  httr::add_headers(Authorization = paste("Key", apiKey)))
```

7 Exercise: Modelling

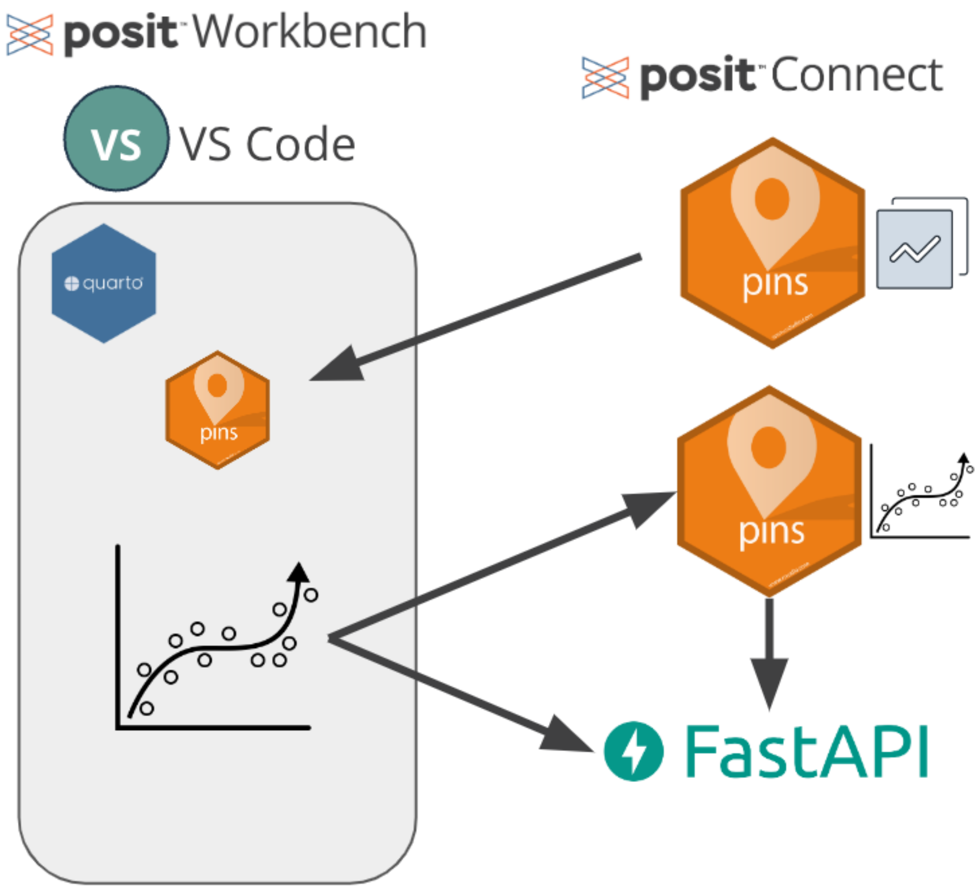


Figure 7.1: Model Workflow

Now that we have a polished dataset pinned to Posit Connect, it's time to *understand* what the data is telling us. Furthermore, we can use the data to train a model to help predict future events. However, **this workshop is not a modeling workshop!** The model we will create in this exercise is a proof-of-concept and will not be particularly informative.

In this exercise (as shown in Figure 7.1), you will:

1. Read in the pinned COVID data

2. Use data to create a linear regression model in Posit Workbench (VS Code)
3. Pin the model to Posit Connect
4. Serve the pinned model as a FastAPI on Posit Connect

7.1 Step 1 - Create a Model

The COVID dataset contains a column (`new_cases`) that shows how many new cases of COVID were reported on a specific day. There is seasonality for certain infections (e.g., influenza and RSV), and we can use this COVID dataset to determine if the same applies to COVID. In other words:

> Can we predict new cases of COVID given a specific day of the year?

To highlight the cross-language functionality of Posit Team, we will create this model using Python! There are so many amazing packages and modules available in Python for creating models including Tensorflow, Keras, and SciKit-Learn. For this workshop we are going to create a simple linear regression model using SciKit-Learn.

7.1.1 Load Necessary Packages

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
```

7.1.2 Read in Pinned COVID Data

Reminder: You'll need to replace `your_name` with your Posit Connect username in all subsequent code blocks.

```python
import pins

board = pins.board_connect(allow_pickle_read=True)

covid = board.pin_read("your_name/covid_data")
```

7.1.3 Transform Data

Before we can model the COVID data with SciKit-Learn, we need to transform it to a format that is conducive to modeling. In the next steps we will convert the date colum to a date *class*, engineer a new feature called `DayOfYear`, and select only the columns of interest (`DayOfYear` and `new_cases`).

```
# Convert to date class
covid["date"] = pd.to_datetime(covid["date"])

# Feature engineering: Extracting day of the year as a feature
covid["DayOfYear"] = covid["date"].dt.dayofyear

# Extract columns of interest
df = covid[["DayOfYear", "new_cases"]]
```

7.1.4 Train Linear Regression Model

```
# Create and train a linear regression model
covid_model = make_pipeline(PolynomialFeatures(4), LinearRegression()).fit(df.drop(columns="
```

7.2 Step 2 - Make Predictions

Let's use our new model to make prediction of how many new cases of COVID there will be given a day of the year. We'll visualize these predictions using `matplotlib`.

Warning

Your predictions/plots may look different depending on which state/province you selected in the previous exercise.

```
# Make predictions
covid_pred = covid_model.predict(df.drop(columns="new_cases"))

# Visualize the results
plt.scatter(df.drop(columns="new_cases"), df["new_cases"], color='black', label='Actual')
plt.scatter(df.drop(columns="new_cases"), covid_pred, color='blue', s=2, label='Predicted')
plt.xlabel('Day of Year')
plt.ylabel('Number of Cases')
plt.legend()
plt.show()
```

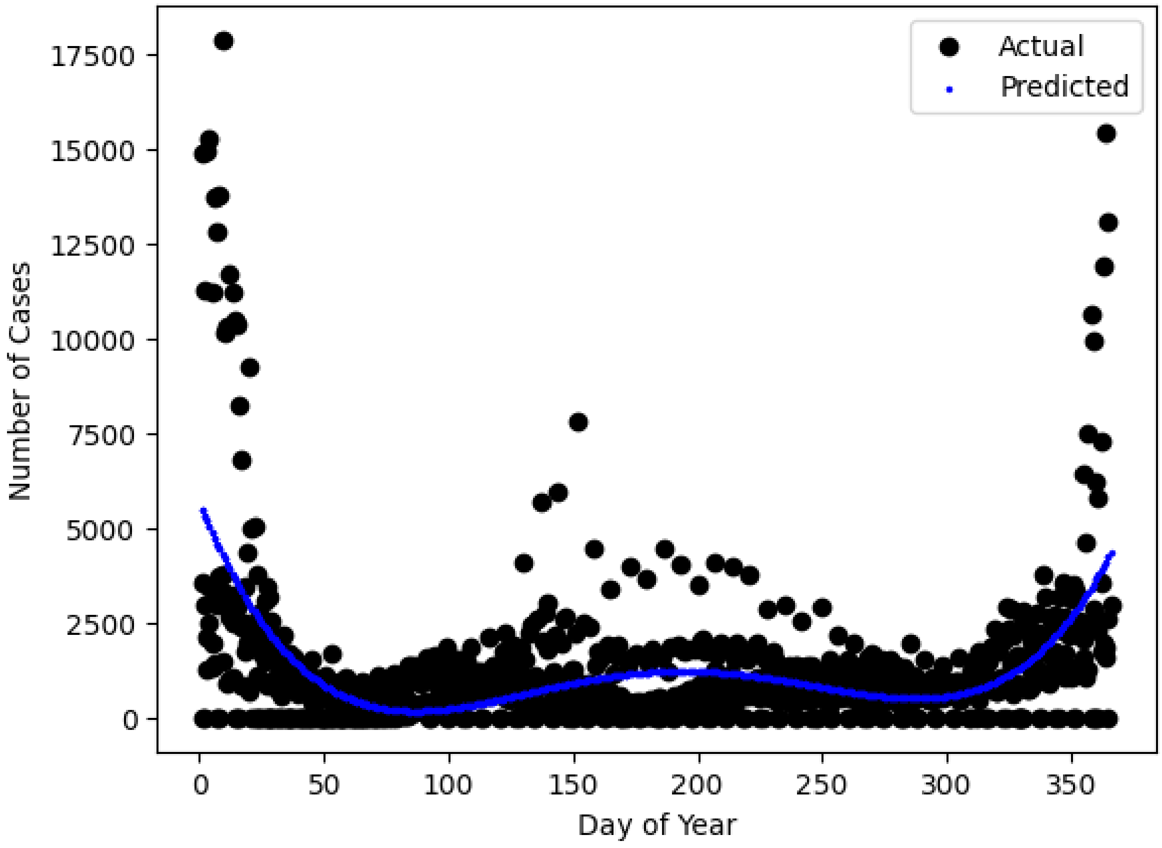


Figure 7.2: New cases of COVID for Maryland

7.3 Step 3 - Save and Serve Model

In the next steps, we'll convert our linear regression model to a *vetiver model* and subsequently save it as a pin to Posit Connect. We'll then use the pinned model to create a FastAPI that will serve our model so we can interact with it.

7.3.1 Create a Vetiver Model

```
from vetiver import VetiverModel

# Create Vetiver model
v = VetiverModel(covid_model, model_name = "publisher1/covid_model", prototype_data=df.drop(
```

7.3.2 Save Model as a Pin to Posit Connect

```
from vertiver import vetiver_pin_write

# Save model as pin to Posit Connect. The "board"" was defined above
vetiver_pin_write(board, v)
```

7.3.3 Serve Model as a FastAPI on Posit Connect

In the below code, we will use the pinned model to serve a FastAPI. For our workshop environment, the `CONNECT_SERVER` and `CONNECT_API_KEY` variables are already present in your environment, and you'll use the `os` package to retrieve them.

```
from vetiver import VetiverAPI, deploy_rsconnect
from rsconnect.api import RSConnectServer

# Define Connect Server
connect_server = RSConnectServer(
    url=os.getenv("CONNECT_SERVER")
    api_key=os.getenv("CONNECT_API_KEY"))

# Deploy FastAPI
deploy_rsconnect(board = board,
                 pin_name = "publisher1/covid_model",
                 connect_serve==connect_server)
```

7.4 Step 4 - Interact with FastAPI

With the FastAPI now deployed to Posit Connect, we can programmatically query the API which will return the predicted number of new COVID cases given a specific day of the year. Before we query the API, there are a few variable we should set including the endpoint as well as the query parameters:

```python
import os
import pandas as pd
from vetiver.server import predict, vetiver_endpoint

# Set vetiver endpoing (URL can be found on Posit Connect)
#  Might need to add "predict" to the end of the URL
endpoint = vetiver_endpoint("https://connectexample.posit.co/cnct/content/{APP_ID}/predict")

# Define API key (if not done already)
api_key = os.getenv("CONNECT_API_KEY")

# Define query parameters. Example: 44th day of the year
params = pd.DataFrame({'DayOfYear': [44]})

# Query API!
h = {"Authorization": f"Key {api_key}"}
predict(endpoint=endpoint, data=params, headers=h)
```

Part IV

Communicate

8 Shiny

You might invest days, weeks, or even months developing an impressive data science workflow and your confident that the results of your workflow are brimming with *value* - value to you, your colleagues, your clients, or decision makers at your company.

Failure to **communicate** the results of your workflow to the pertinent parties leads to a loss of value. As such, it's worth investing in strategies that can make your workflow results "shine bright" for all to see and consume. Enter `shiny`!

Shiny is an open-source R and Python package and is used to build *interactive web applications*. The best part? You don't have to be a web developer to create a shiny app! You just need to know a bit of R and/or Python. Shiny is 100% code-based, which sets it apart from other popular business intelligence (BI) tools.

8.1 What's in a Shiny App?

Most shiny applications have 4 components:

- **Header:** The header is usually at the very top of the shiny application. This is a useful space for things that need to be executed as soon as someone opens your shiny application such as reading in any data or packages.
- **User Interface (UI):** The UI is where design the visual layout of your application. Specifically where to put your inputs and outputs.
- **Server Function:** The server function tells shiny *how* to build the various outputs.
- **Run app:** This is where you combine the UI and Server function into a single shiny application.

Below is the code for a simple shiny application, written in both R and Python:

8.2 Python

```python
# Header
from shiny import *

# UI
app_ui = ui.page_fluid(
    ui.input_slider("n", "N", 1, 100, 40),
    ui.output_text_verbatim("txt"),
)
```

```
# Server Function
def server(input, output, session):
    @output
    @render.text
    def txt():
        return f"The value of n*2 is {input.n() * 2}"

# Run App
app = App(app_ui, server)
```

8.3 R

```
# Header
library(shiny)

# UI
ui <- fluidPage(
  sliderInput("n", "N", 1, 100, 40),
  textOutput("txt")

)

# Server Function
server <- function(input, output) {
  output$txt <- renderText({
    paste0("The value of n*2 is: ", input$n * 2)
  })
}

# Run App
shinyApp(ui = ui, server = server)
```

8.4 Incorporating Shiny into your Workflow

It can be tempting to incorporate your *entire* data science workflow into your shiny application. This often leads to bloated applications that are frustratingly slow (see left side of Figure 8.1). Instead, it's recommended to decouple the "computationally heavy" parts of your workflow from your shiny application where possible (see right side of Figure 8.1). Consider the two examples below (both of which we've discussed in previous chapters):

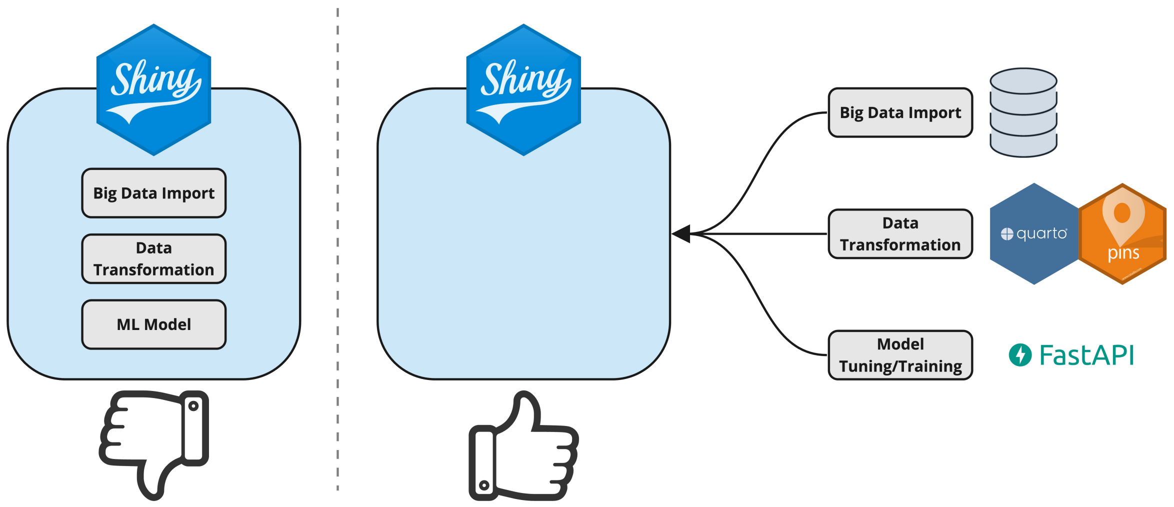


Figure 8.1: Shiny application best practices

- If your shiny application relies on large datasets, make sure to only read in the data required for the application. Also consider pre-processing your data before reading it into the shiny application.
- If your shiny application leverages a machine learning model, consider training/tuning your model outside of shiny and serving it as an API.

By keeping the steps of your workflow independent from each other, you can dramatically improve your shiny application's performance and simplify the underlying code. These are two big steps for making your shiny applications more **production grade!**

9 Exercise: Shiny for Python

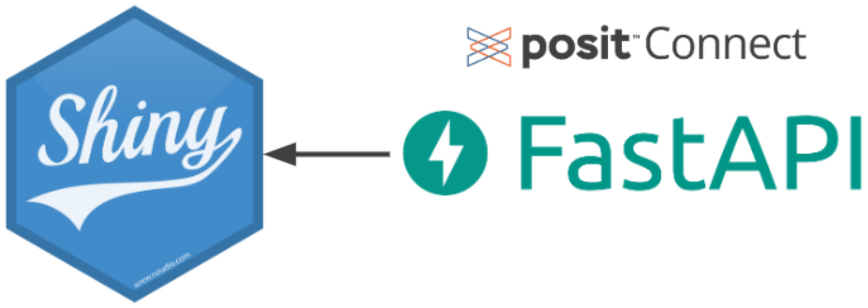
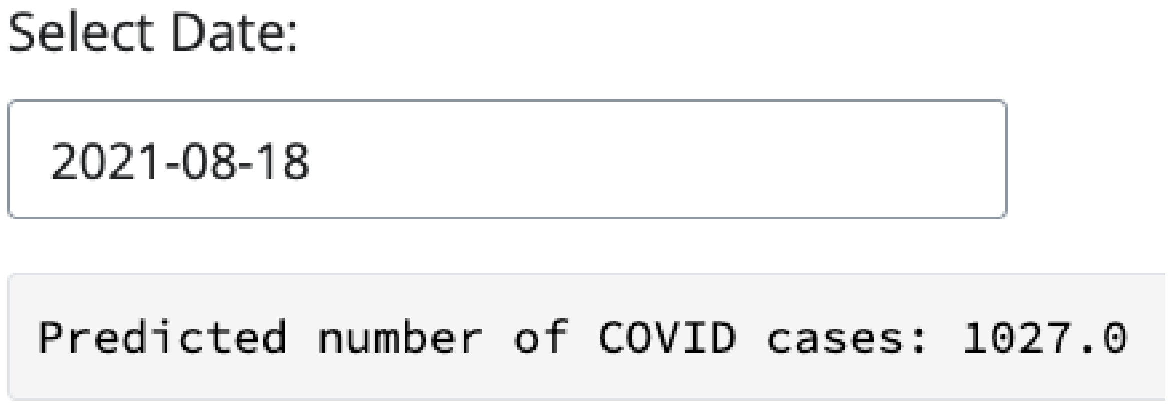


Figure 9.1: Shiny + FastAPI workflow

In this exercise, you will get practice creating a Shiny or Python application in Posit Workbench (VS Code) that makes predictions of new cases of COVID given a specific day of the year. This shiny application will be fairly "lightweight" in that it's primary role is to query the FastAPI on Posit Connect that serving our COVID model we created earlier in Chapter 7. The user interface will only contain two things: a date selection box, and a text box with the predicted number of COVID cases:


Select Date:

2021-08-18

```
Predicted number of COVID cases: 1027.0
```

Figure 9.2: COVID Prediction App - Shiny for Python

9.1 Step 1 - Set the Stage

Before we define our Shiny app user interface and server function, we first need to import the necessary packages and define some variables.

9.1.1 Import Necessary Packages

```
from shiny import App, render, ui
import os
import vetiver
import pandas as pd
```

9.1.2 Set Variables

The primary variable we need to set is the vetiver endpoint. You'll need to supply the API URL which can be found on Posit Connect. The other variable is the Posit Connect API key, which is only needed if your API requires authenticated access.

```
# Define endpoint for API and key
endpoint = vetiver.vetiver_endpoint("https://connectexample.posit.co/cnct/content/{APP_ID}/p

api_key = os.getenv("CONNECT_API_KEY")
```

9.2 Step 2 - Create User Interface

As shown in Figure 9.2, our application only has an *input date*, and *output text*.

```
# User Interface
app_ui = ui.page_fluid(
    ui.input_date("day", "Select Date:", value="2021-01-01"),
    ui.output_text_verbatim("txt")
)
```

9.3 Step 3 - Create Server Function

The goal of the server function is to query the FastAPI and use the API response to create the text output. After the server function, be sure to call the `App` function to bring together our UI and server function into a single app object!

```python
# Server Function
def server(input, output, session):
    @render.text
    def txt():

        # Parameters to be included in the query string
        #  Convert date to number of year
        params = pd.DataFrame({
            'DayOfYear': [input.day().strftime("%j")]
        })

        # If needed, add authorization
        h = {"Authorization": f"Key {api_key}"}

        # Make a prediction
        response = vetiver.predict(endpoint=endpoint, data=params, headers=h).at[0, "predict

        # Return message
        return f"Predicted number of COVID cases: {response}"

# Create Shiny App
app = App(app_ui, server)
```

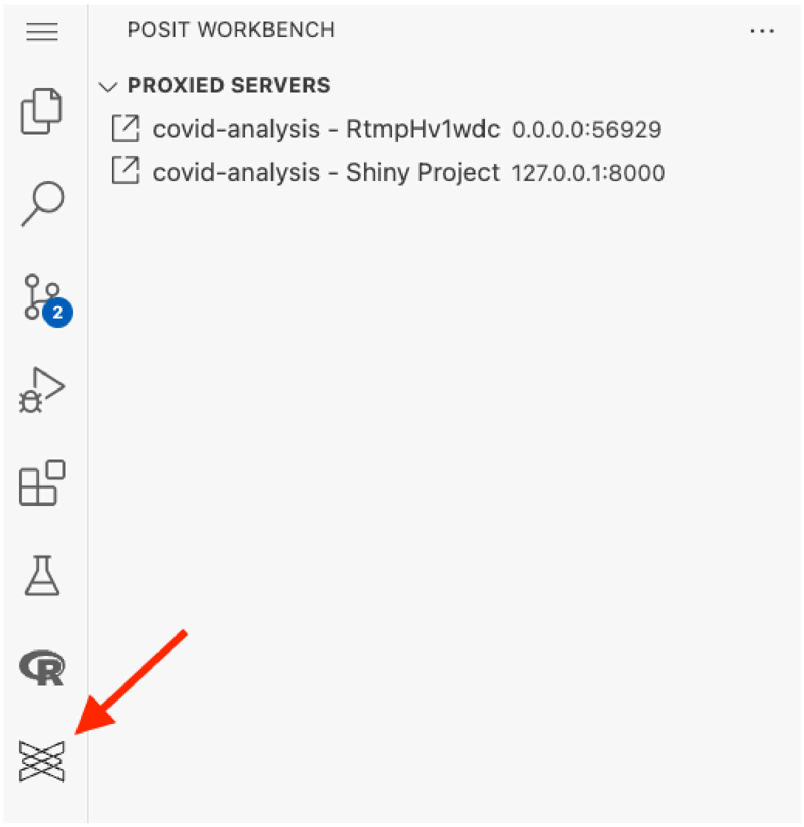
9.4 Step 4 - Run the Application on Posit Workbench

Access the terminal within VS Code (click the three line icon in the top left corner followed by `Terminal` –> `New Terminal`). The workshop environment is already configured with the `shiny` CLI tool which we can use to run the shiny application.

To run the shiny application, make sure you've saved your application as `app.py` and you know where it's located in your file system, then type the following command into the terminal:

```
shiny run /path/to/app.py
```

You'll see a link appear in the terminal that you can click, or you can use the Posit Workbench VS Code extension (red arrow in Figure 9.3) to view actively running "proxied servers." In the below example, you can see two servers running. The bottom server is actively running our shiny app!



covid-analysis - RtmpHv1wdc 0.0.0.0:56929
covid-analysis - Shiny Project 127.0.0.1:8000

Figure 9.3: Posit Workbench VS Code Extension

9.5 Step 5 - Publish Shiny Application to Posit Connect

In the last step, we'll show how to use the `rsconnect-python` CLI tool to publish our shiny for python application to Posit Connect.

The only prerequisite needed, is to define the instance of Posit Connect you would like to publish to. In the workshop environment, this has already been configured, and you can run `rsconnect list` in the terminal to view the details. See Posit's documentation for how to add a new Posit Connect instance if needed.

To publish the shiny application, run the following command in the terminal:

```
rsconnect deploy shiny path/to/app.py/
```

The only argument you need to supply is the directory that contains the shiny for python application (called `app.py`). Once the deployment process is complete, you can click the links at the bottom of the terminal (see example terminal output below) to view the application now hosted on Posit Connect!

```
Deployment completed successfully.
         Dashboard content URL: https://connectexample.posit.co/cnct/connect/#/{APP_ID}
         Direct content URL: https://connectexample.posit.co/cnct/content/{APP_ID}/
```

10 Exercise: Shiny for R

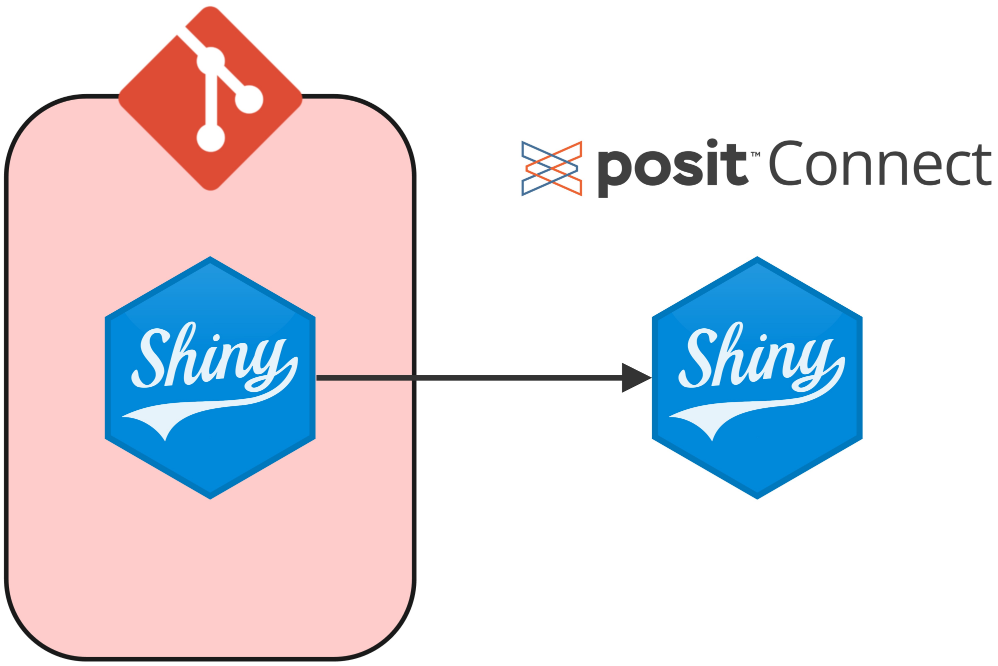


Figure 10.1: Git Backed Deployment Workflow

In the previous exercise (Chapter 9), we created a shiny application and **manually** deployed it to Posit Connect. What happens if you make a change to your application? Well you'll have to manually deploy again…and again…and again.

Automating manual processes can save your team time and headaches. In this exercise, you will get practice automating the deployment process by using git-backed deployment!

10.1 Introduction to Git-Backed Deployment

Every data science workflow should use version control! The most popular version control system is *git*, which allows you to track the evolution of your workflow's source code. You can host the tracked files in what's known as a *repository.*

If the content your working on (or maybe collaboratively working on) lives withing in git repository, then you can publish it directly to Posit Connect! The only requirement is you'll need to supply a companion `manifest.json` file for your content. This file contains information about the development environment so that Posit Connect knows which packages, package version, and R/Python versions are needed to run the content on Posit Connect. To create a manifest file for your R/Python content, see the documentation here and example code snippets below:

10.2 Python

```
rsconnect write-manifest
```

10.3 R

```
library(rsconnect)

writeManifest()
```

Once a piece of content is deployed to Posit Connect via git-backed deployment, anytime there is an update to the content within the git repository, Posit Connect will detect that change and **automatically redeploy the content!**

For this exercise, we've created a GitHub repository that contains a Shiny application (built using R) which we'll deploy to Posit Connect.

10.4 Step 1 - Initiate Git-Backed Deployment on Posit Connect

Within the GitHub repository is a directory called `shiny-app-r`. Within that directory will be the two files required for git-backed deployment: the application itself (`app.py`) and the required `manifest.json` file.

Within Posit Connect, navigate to the **Content** page and click the **Publish** button at the top. In the dropdown menu, select **Import from Git**.

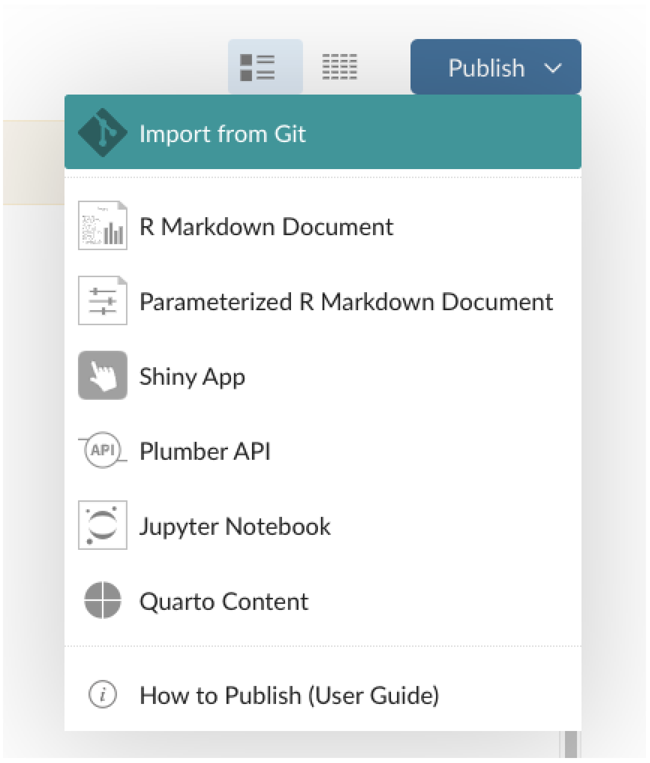
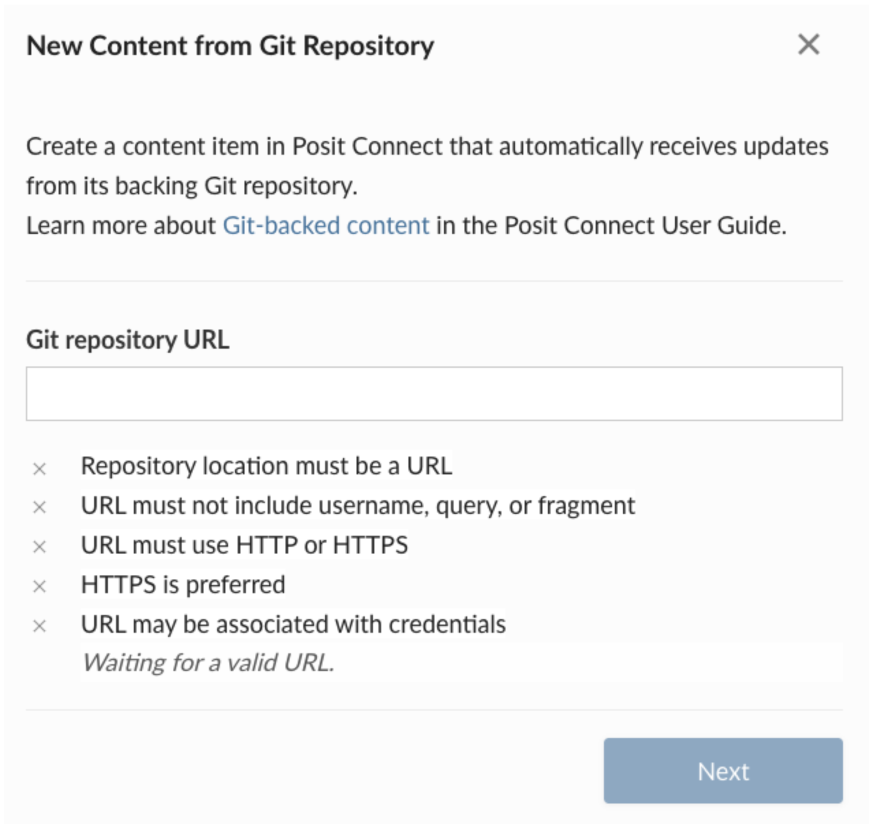


Figure 10.2: Publish Drop Down Options


New Content from Git Repository

Create a content item in Posit Connect that automatically receives updates from its backing Git repository.
Learn more about Git-backed content in the Posit Connect User Guide.

Git repository URL

× Repository location must be a URL
× URL must not include username, query, or fragment
× URL must use HTTP or HTTPS
× HTTPS is preferred
× URL may be associated with credentials
Waiting for a valid URL.

Figure 10.3: Git Repository URL

10.5 Step 2 - Add Repository Details

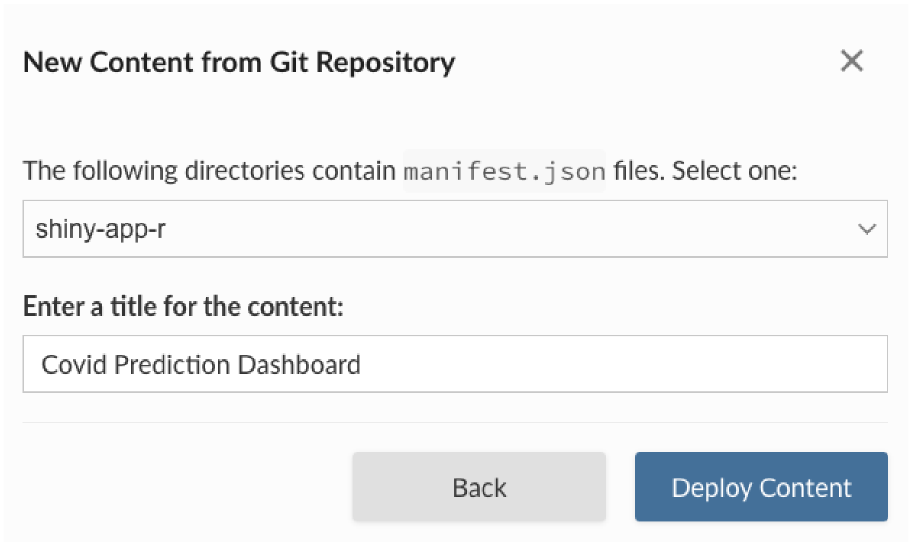
In the next popup window, you'll need to supply the URL for the git repository.

After supplying the URL, click **Next** where you'll be asked to select a branch. This repository only contains a single branch called `main`. Ensure `main` is selected and click **Next**.

> Tip
>
> You can deploy the same piece of content from a single git repository! For example, maybe you have a developmental version on a `dev` branch, and a production version on the `main` branch.

10.6 Step 3 - Deploy Content

At this point, Posit Connect will look for "deployable directories." In other words, it's looking for directories in the repository that contain a `manifest.json` file. There should only be one deployable directory (`shiny-app-r`). Ensure it's selected, give your application a name, then click **Deploy Content**!


New Content from Git Repository

The following directories contain `manifest.json` files. Select one:

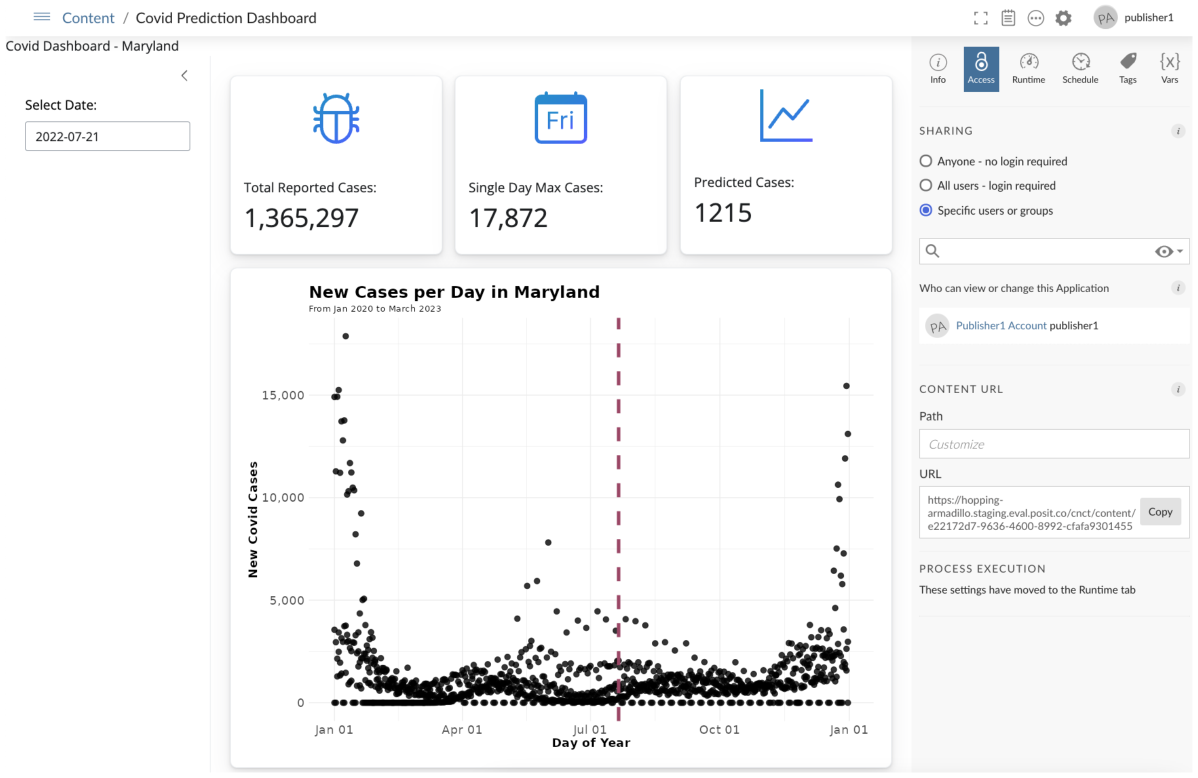
shiny-app-r

Enter a title for the content:

Covid Prediction Dashboard

Figure 10.4: Deploy Content from Git Repository

Connect will then read the `manifest.json` file and install any environment dependencies needed to ensure the application runs properly. After the deployment process is complete, click **Open Content** to view the application on Posit Connect!



Content / Covid Prediction Dashboard

publisher1

Covid Dashboard - Maryland

Select Date:

2022-07-21

Total Reported Cases:
1,365,297

Single Day Max Cases:
17,872

Predicted Cases:
1215

SHARING

Anyone - no login required
All users - login required
[✓ Specific users or groups]

Who can view or change this Application

Publisher1 Account publisher1

CONTENT URL

Path

Customize

URL

https://hopping-armadillo.staging.eval.posit.co/cnct/content/e22172d7-9636-4600-8992-cfafa9301455 Copy

PROCESS EXECUTION

These settings have moved to the Runtime tab

Figure 10.5: Covid Dashboard

This shiny dashboard tells us the same information as our previous shiny for Python application: how many new COVID cases are predicted for a given day of the year. As you can see, this application has some additional details and features that greatly improve the user experience including the total number of COVID cases for the given state/province, and the single day max number of new COVID cases.

By default, Posit Connect will scan the linked git repository for any changes every 15 minutes. If any changes were detected (e.g., a new commit to the `main` branch), Posit Connect will automatically redeploy the application! You can also manually check for repository updates by clicking the **Info** tab in the top right corner and selection **Update Now** from the side bar:

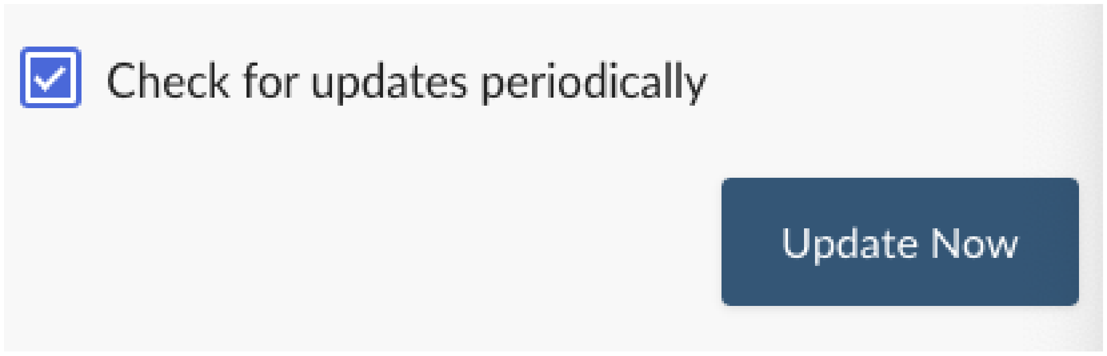


Figure 10.6: Manually Check Git Repository for Changes

Part V

Bonus Content

11 Bonus Content