Center for Education Policy Research
HARVARD UNIVERSITY

STRATEGIC DATA PROJECT
2026 CONVENING

PROOF POSITIVE

Making Data Interactive with R Shiny

A Teacher Workforce Planning Example

Key Objectives

- Understand the basic structure of a Shiny app
- Make interface and layout changes
- Explore how the user interface connects to app logic
- Add new outputs that display existing data
- Add user controls that update what the app shows

Session Norms

- Stop me anytime if something does not make sense.
- No bad questions, especially with Shiny.
- Confusion is expected and welcomed.
- Mistakes and errors are where the learning happens.


Shiny
BUILT DIFFERENT FOR SMART PEOPLE
DATA FUEL
CONTROLS
HORIZON
24M
GROWTH
1.8x
PRICE
SEASONALITY
RISK TOLERANCE
RUN MAGIC
FORECAST
Now
6M
12M
18M
24M
SCENARIOS
BEST CASE
BASE CASE
STRETCH
NOPE CASE
MODEL LAB
MODEL
XGBoost v3.14
MAPE
6.7%
FEATURES
42
READY
FEATURE IMPORTANCE
I DON'T CLICK BUTTONS. I SHIP INSIGHT.
purr-formance optimized
INSIGHT LEVEL
INSIGHT THROTTLE
DO EPIC STUFF
CAFFEINE + DATA
Custom app energy


Power BI / Tableau
FILTERS
Date
Region
Business Unit
Product
Customer Segment
Revenue
$24.8M
Profit
$6.7M
Margin
27.0%
YoY Growth
18.6%
Revenue Over Time
Revenue by Region
Top Products
Revenue by Segment
Actual vs Target
118%
GOVERNED & APPROVED
ONE SOURCE
ONE TRUTH
ONE DASHBOARD
DASHBOARD FACTORY
Dashboard at scale

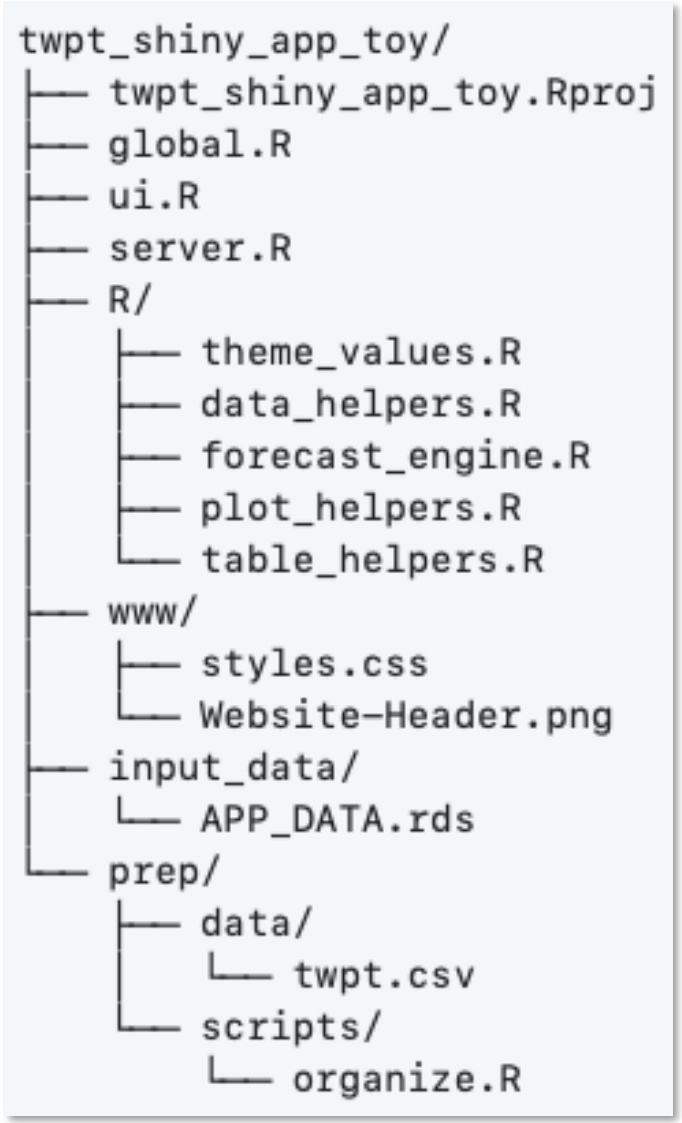
Teacher Workforce Planning Tool

https://mattfaiello.shinyapps.io/TWPT/

What You Need

- ✓R installed
- ✓RStudio installed
  - ➢https://posit.co/download/rstudio-desktop
- ✓Workshop repository downloaded from GitHub
  - ➢https://github.com/MatthewFaiello/twpt_shiny_app_toy

Download the twpt_shiny_app_toy Repository



Open twpt_shiny_app_toy.Rproj

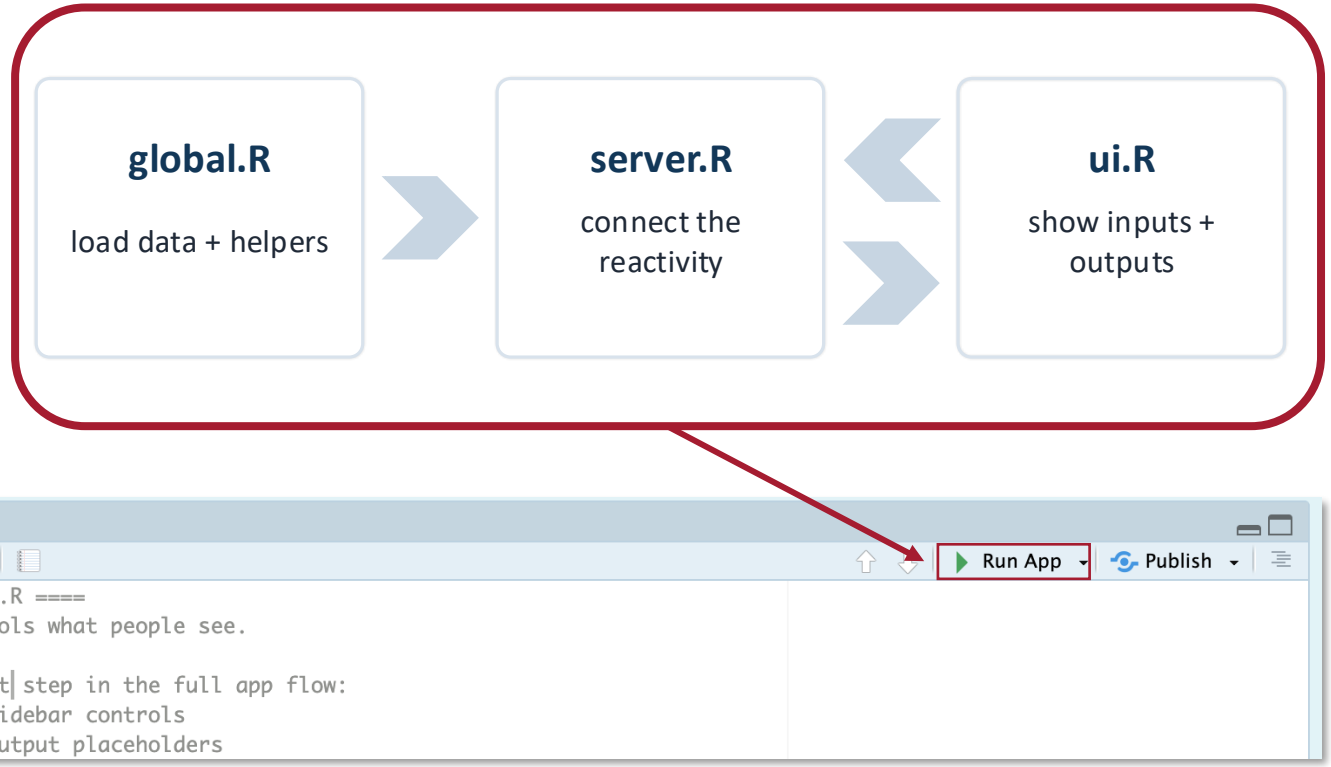
It will open RStudio inside the R project directory.

Open prep/scripts/organize.R

Data prep: read, clean, compute, save

Core Scripts

Keep in the same directory



Open ui.R

What users see

UI Layout: Shell → Tabs → Panels

page_sidebar() = controls on the side, results in the main area
navset_card_tab() = tab container
nav_panel() = one labeled tab and its contents

What it is

Layout functions define how the app is arranged on the page.

Why it matters in your app

The filters stay fixed in the sidebar while users switch between the trend plot and underlying data table.

In the toy app

```
ui <-
  page_sidebar(
    sidebar = sidebar(
      selectizeInput(...),
      selectizeInput(...),
      downloadButton(...)
    ),

    navset_card_tab(
      nav_panel(
        LABELS$tab_plot,
        textOutput("scope_note"),
        plotOutput("main_plot")
      ),

      nav_panel(
        LABELS$tab_data,
        DTOutput("detail_table")
      )
    )
  )
```

Think of it as the overall page shell for the app.

Hands-on Break 1

Rearrange the UI (10 minutes)

Try…

- Moving the download button higher or lower in the sidebar.
- Add text the sidebar.
- Add a blank About tab with static text.

Success check

- The app still runs.
- The plot still appears.
- No other edits were needed.

In the toy app

```
# find first `<- Code here`
# add this to sidebar
h4("Display")

# find second `<- Code here`
# add this to navset_card_tab()
nav_panel(title = LABELS$tab_data)
```

Think of it as the overall page shell for the app.

Debrief 1

What changed, and what stayed the same?

Open global.R

Contains packages, labels, data, choices, defaults, and helper functions

Creating an R Function

function() = define a reusable tool.

What it is

A function is a reusable block of code. You give it inputs, it does a job, and it returns a result.

Why it matters in your app

In your app, helper functions keep repeated logic out of server.R. That makes the app easier to read, reuse, and explain.

In the toy app

```
data_filtered <-
  function(data = APP_DATA,
           scope_1 = DEFAULTS$scope_1,
           scope_2 = DEFAULTS$scope_2) {

    data %>%
      filter(County_Name == scope_1,
             lea == scope_2) %>%
      arrange(SchoolYear)
  }
```

This helper returns the rows for the selected county and LEA.

Open server.R and ui.R

Now we connect

Reactivity

reactive() = compute a live value.
req() = do not continue until values are ready.

What it is

A reactive expression builds a value that updates automatically when its inputs change.

Why it matters in your app

Your app uses reactive() for both the LEA choice list and the central filtered dataset. That lets several outputs reuse the same up-to-date data.

In the toy app

```
filtered_data <-
  reactive({
    req(input$scope_1, input$scope_2)

    data_filtered(data = APP_DATA,
                  scope_1 = input$scope_1,
                  scope_2 = input$scope_2)
  })
```

This becomes the main dataset for the plot, note, table, and download.

Outputs

*Output() = reserve space in UI.
output$... <- render*() = fill space in server.

What it is

*Output() functions live in ui.R and reserve a spot on the page. output$ lives in server.R and fills that spot with actual content.

Why it matters in your app

The ids must match. If they do not match, the page has a placeholder with nothing connected to it.

In the toy app

```
# ui.R
textOutput("scope_note")
plotOutput("main_plot")
DTOutput("detail_table")
downloadButton(outputId = "download_data")

# server.R
output$scope_note <- renderText({...})
output$main_plot <- renderPlot({...})
output$detail_table <- renderDT({...})
output$download_data <- downloadHandler(...)
```

Match the ids exactly: scope_note, main_plot, detail_table, download_data.

Rendering Outputs

render*() = build the thing.
*Output() = show where it goes.

What it is

render*() functions tell Shiny how to build something visible: a plot, text, a table, and more.

Why it matters in your app

They connect your reactive data to the placeholders in ui.R. Each render function sends one finished output back to the page.

In the toy app

```
output$main_plot <-
  renderPlot({
    dat <- filtered_data()
    plot_staffing_trend(dat)
  }, res = 125)

output$scope_note <-
  renderText({
    dat <- filtered_data()
    make_scope_note(dat)
  })
```

The plotting and text-building logic can live in helper functions from global.R.

Hands-on Break 2

Connect a table to the ui (10-15 minutes)

Context

- selected_table_data() is already available, and the downloader is already using it.
- It's a reactive object that formats the forecast data for the table and download.

Success check

- The Data tab shows the same rows that download into the CSV file.

In the toy app

```
selected_table_data <-
  reactive({
    forecast_table_data(
      data = selected_forecast_data()
    )
  })

# download_data already uses
selected_table_data()
```

Think of it as the overall page shell for the app.

Break 2 Code

server.R

```
# find only `<- Code here`
output$detail_table <-
  renderDT({

    forecast_dt(
      data = selected_table_data()
    )
  })
```

ui.R

```
# add to break 1
nav_panel(
  LABELS$tab_data,
  DTOutput("detail_table")
)
```

Debrief 2

Output() reserves a place. output$... <- render() fills it.

Inputs

*Input() = make the widget.
input$<id> = read the current value.

What it is

*Input() functions create widgets in ui.R. input$<id> is how server.R reads the current value the user picked or typed.

Why it matters in your app

This is the core flow into the server: the widget exists in the UI, and its current value arrives as input$<id>.

In the toy app

```
# ui.R
selectizeInput(inputId = "scope_1",
               label = LABELS$scope_1,
               choices = SCOPE_1_CHOICES)

# server.R
APP_DATA %>%
  filter(County_Name == input$scope_1)
```

The inputId is the contract. If the UI says scope_1, the server reads it as input$scope_1.

Updating

reactive() computes a value.
observeEvent() performs an action.

What it is

observeEvent() responds to a change and performs an action. Unlike reactive(), it is not mainly for returning a reusable value.

Why it matters in your app

In your app, when the county changes, the LEA dropdown needs new choices. That is an action pushed into the interface.

In the toy app

```
observeEvent(input$scope_1, {
  choices <- lea_choices()

  updateSelectizeInput(session = session,
                       inputId = "scope_2",
                       choices = choices,
                       selected = choices[1],
                       server = TRUE)
})
```

The event is the county changing; the action is updating the LEA widget.

Hands-on Break 3

Target change: replace the fixed metric with input$metric in both places. (15 minutes)

Context

The chart and the sentence should reference the same user input:

1. Add the input in the sidebar.
2. Use input$metric in the plot output.
3. Use input$metric in the scope note output.

Success check

- Run the app.
- Switch metrics.
- Watch plot change.

In the toy app

```
plot_staffing_forecast(
  data = selected_forecast_data(),
  metric = "population"
)

make_scope_note(
  data = selected_forecast_data(),
  metric = "population"
)
```

Think of it as the overall page shell for the app.

Break 3 Code

server.R

```
# replace metric = "population" w/
metric = input$metric
```

ui.R

```
# below h4("Display"), from break 1

# add this
selectInput(
  inputId = "metric",
  label = "Metric to plot",
  choices = c(
    "Population" = "population",
    "Matriculation" = "matriculation",
    "Students per Teacher" =
"StudentsPerTeacher",
    "Student Total" = "studentTotal",
    "Teacher Total" = "teacherTotal"
  ),
  selected = "population"
),
```

Debrief 3

The full input contract is now in place.

Questions

Anything that came during the hands-on breaks.

Key Questions Before Building

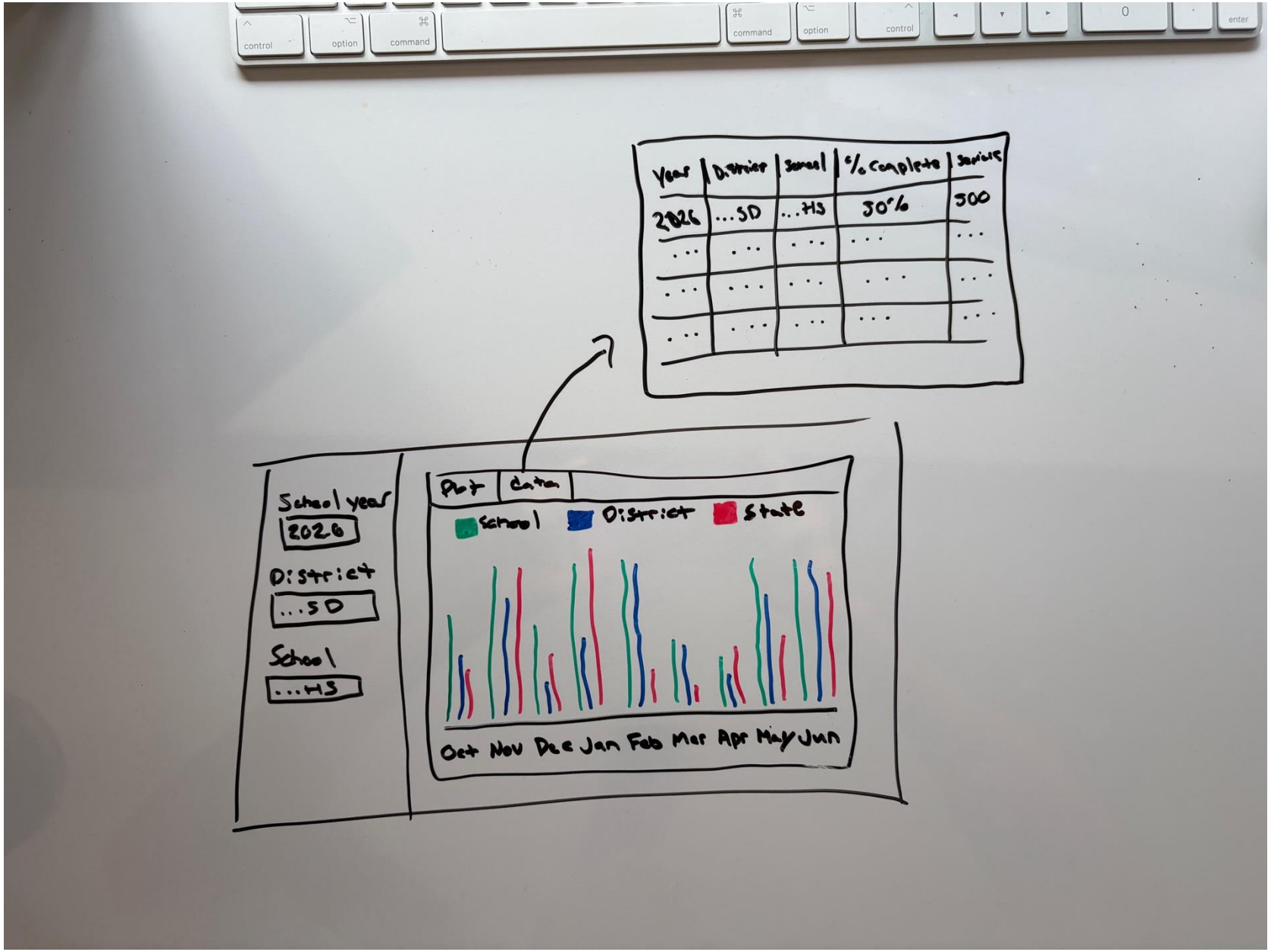
- **Understand the data.** What is being measured, and why does it matter?
- **Understand the users.** Who needs this information, and how are they using it now?
- **Understand the opportunity.** How could a Shiny app make the data easier to access, interpret, and act on?

User Story

As a…

I want…

so that…


Year
% Complete
2026
...SD
...HS
30%
300
School year
2026
District
...SD
School
...HS
Plot
Data
School
District
State
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun

Thank You

matthew.faiello@doe.k12.de.us