Bellwether

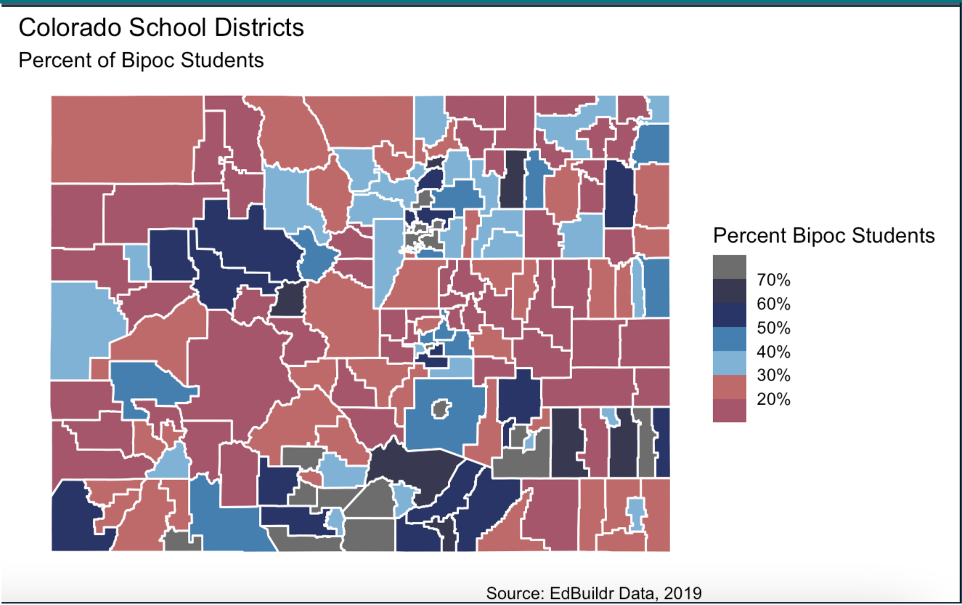
Education Finance Equity Training
Course 2, Class 3

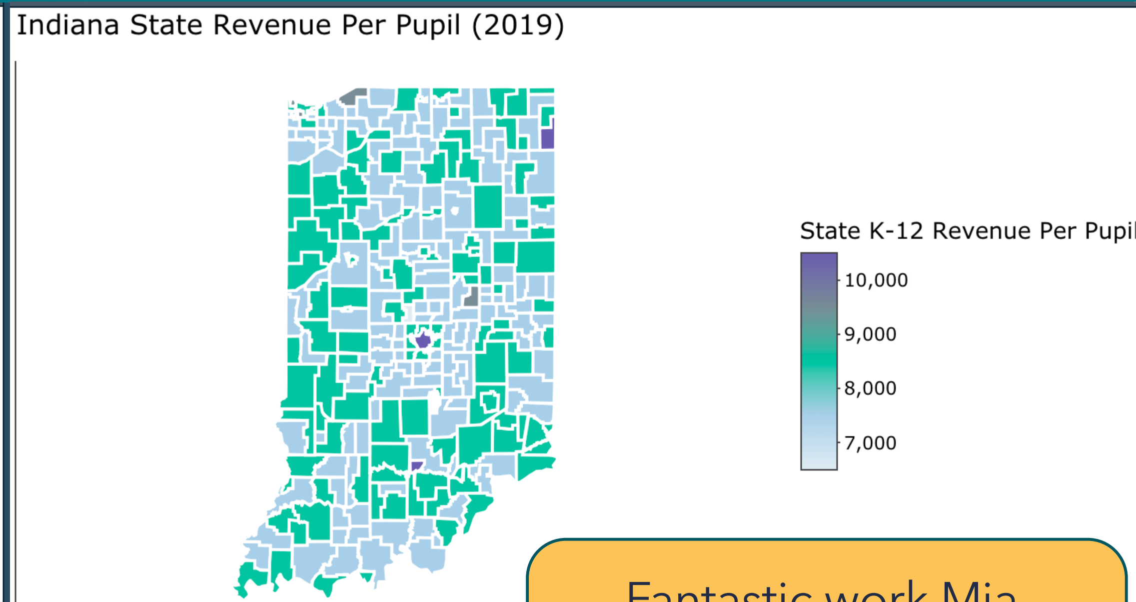
10.04.2022

Agenda

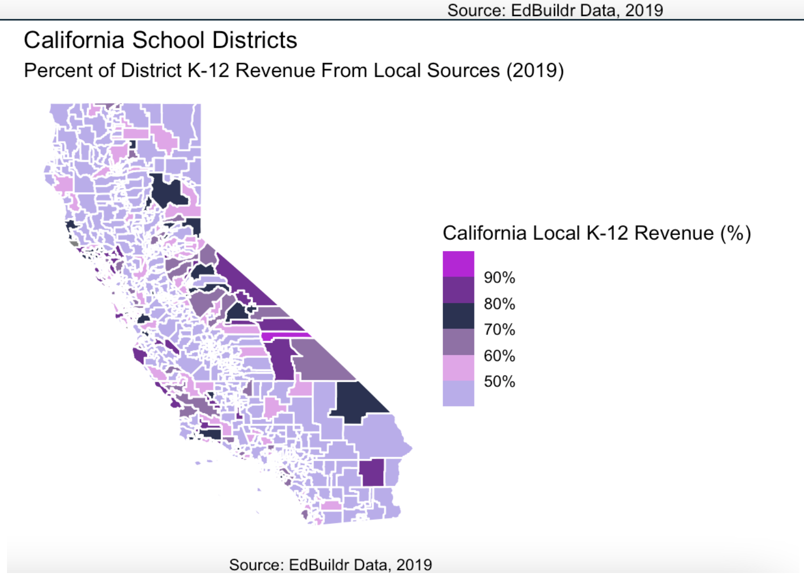
Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

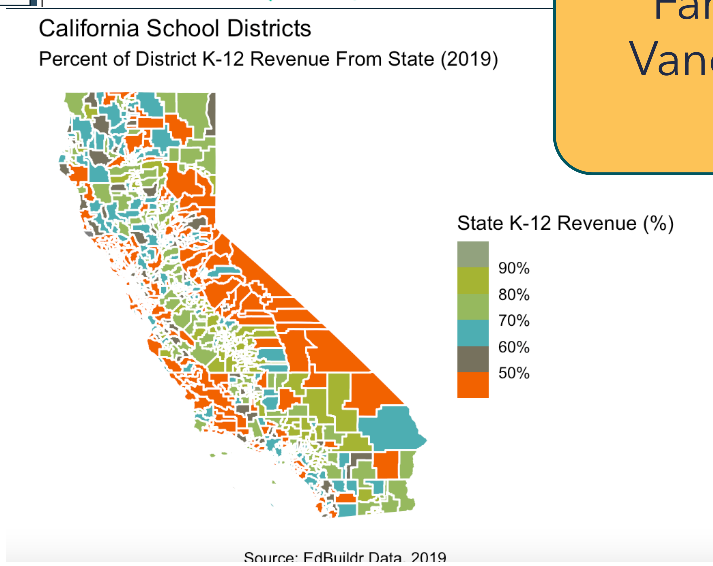
Overview of Homework: Great job on your maps!!!





Fantastic work Mia, Vanessa, Christa, and Jared!





Bellwether

If you're not following #rstats on Twitter, you're missing out on a lot of great content

ggplot is a little bit like cake…

We *always* start by setting up the foundation with **ggplot()**

We specify our ingredients (data variables) with an **aes mapping**

We can create *layers* to our plot with **geoms**

We can style our ~~cake~~ ggplot with **themes.** We have out-of-the-box options, or we can go totally custom!

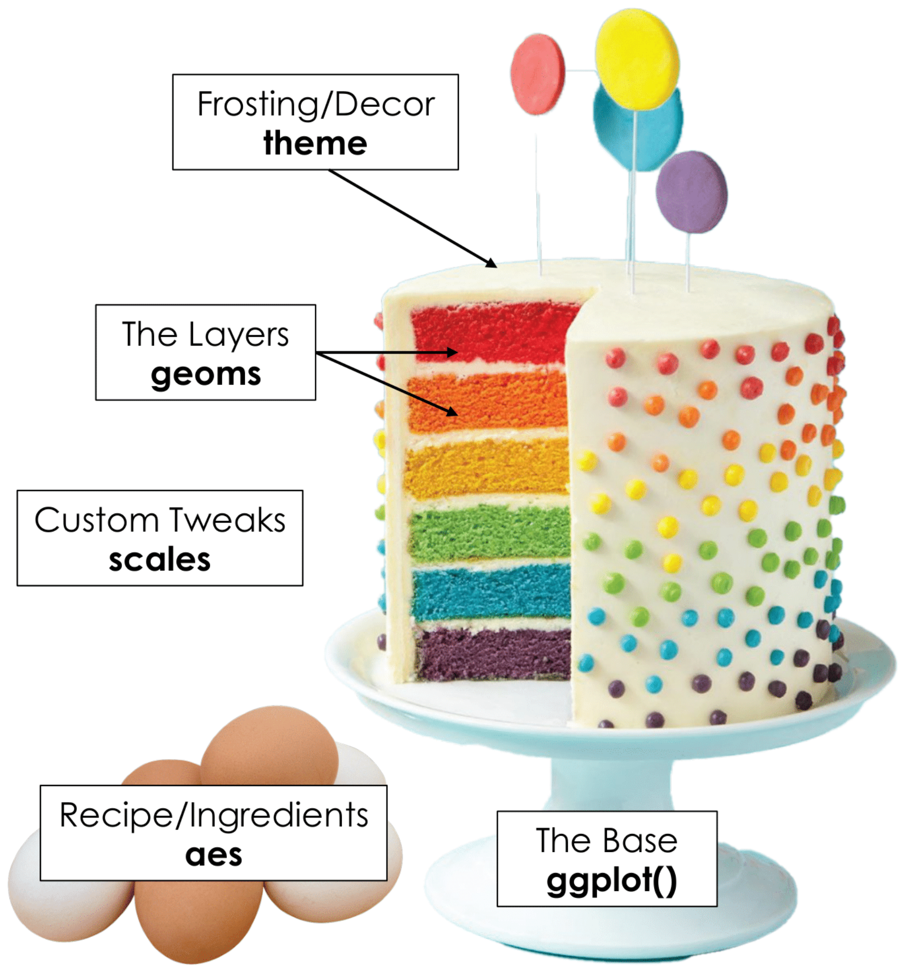


Image from @tanya_shapiro

If you're not following #rstats on Twitter, you're missing out on a lot of great content

Arthur Welle
@ArthurWelle

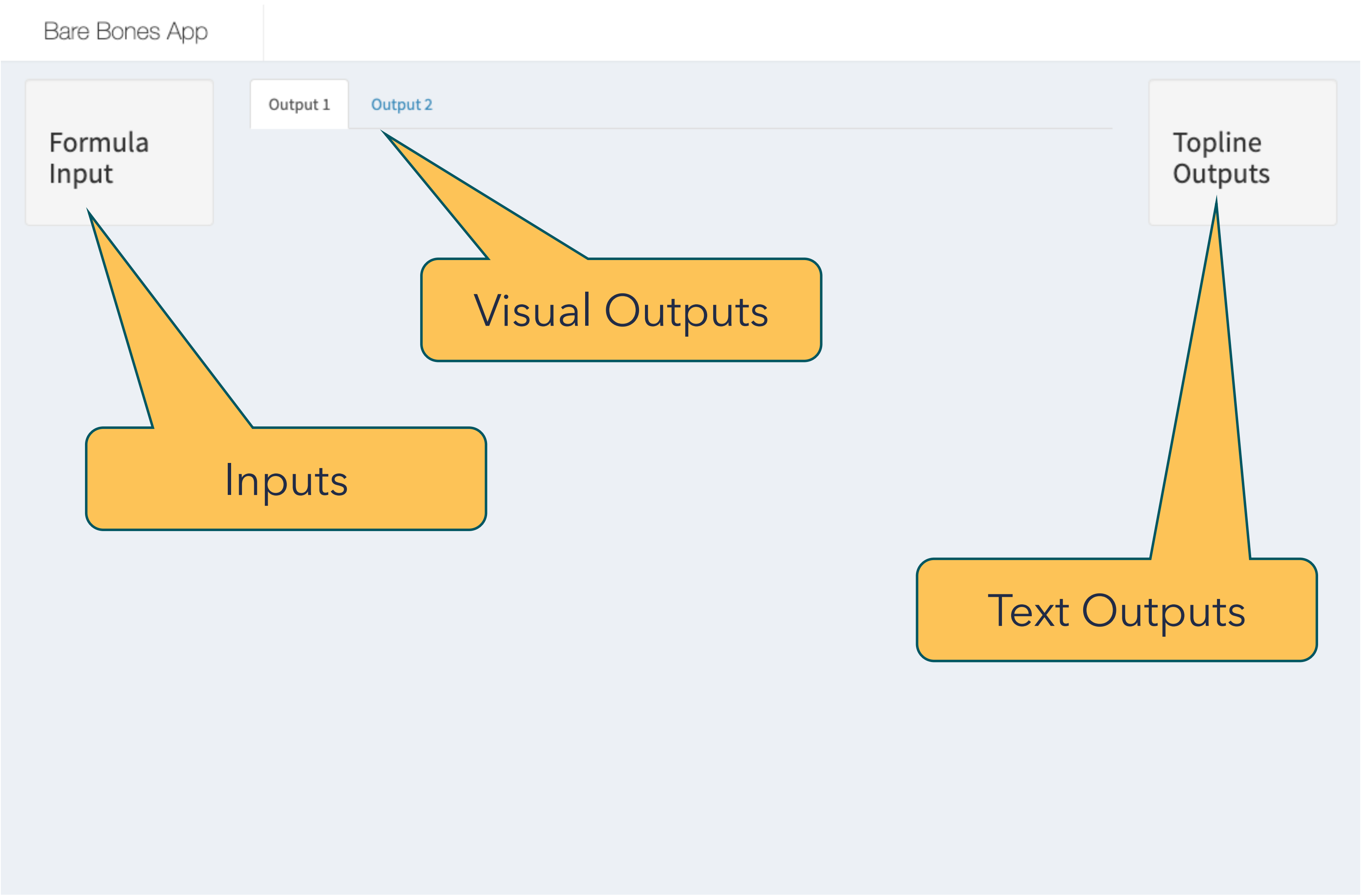
Pipes in #RStats! Functions are verbs, aplied to objects. With pipes "|>" we arrange functions in the order that we would think of the actions. Readability for the win! (And I like to read the pipe as "and then...") 🎂🥣🥧🧑‍🍳🔪🍴

GIF from [@ArthurWelle]

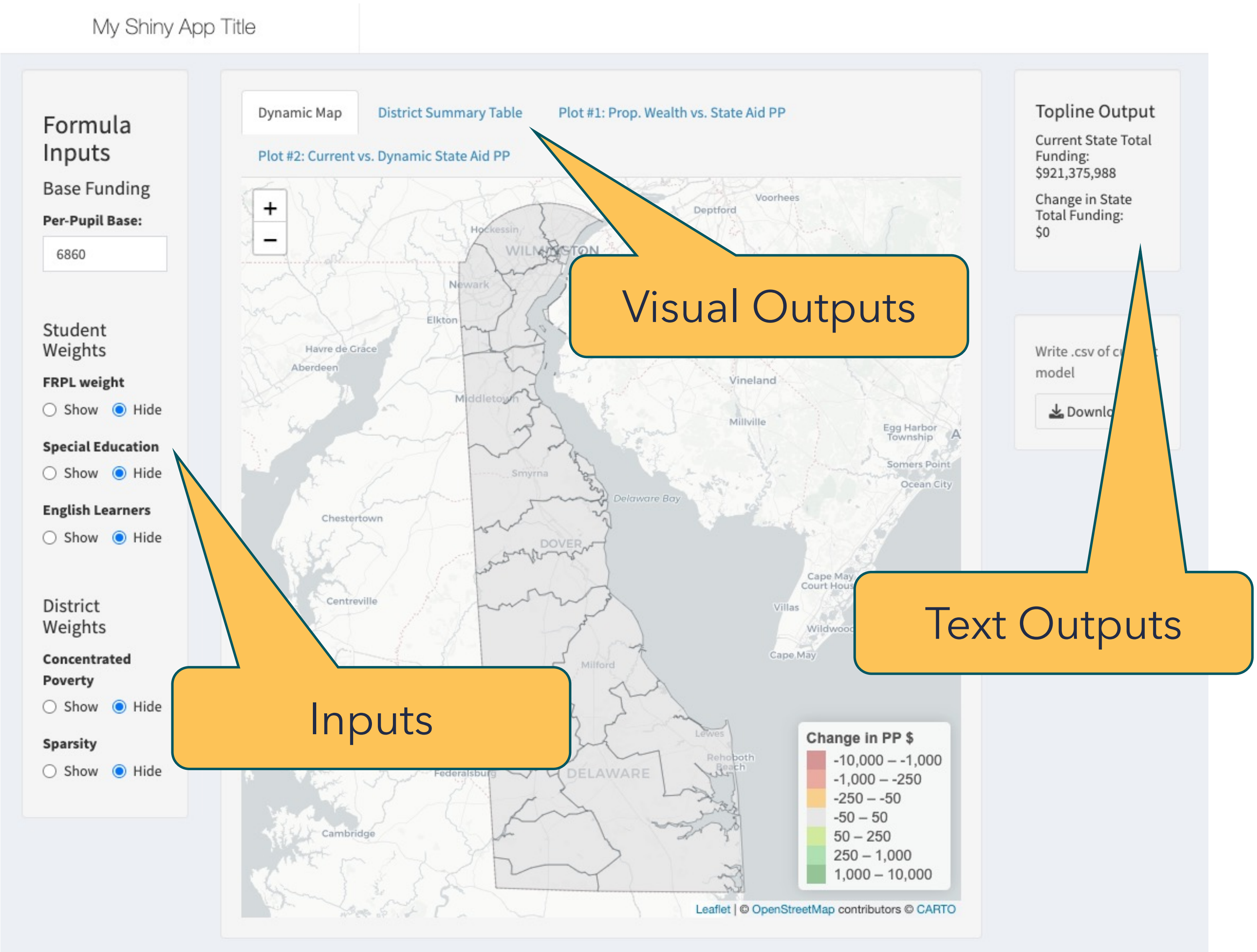
Agenda

Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

Our school finance formula simulators will be built on a simple, but powerful framework: shiny dashboards



By slowly and incrementally updating your dashboard, you can build a powerful tool for school funding analysis and advocacy



Let’s begin by taking a look under the hood of our bare-bones example of a shiny dashboard

```r
shinyUI({

  header <- dashboardHeader(title = "Bare Bones App")

  body <- dashboardBody(
    fluidPage(
      theme = "yeti",
      fluidRow(
        column(2,
               wellPanel(
                 h3("Formula Input")
                 ) # close well panel
               ), # close input column


        column(8,
               tabsetPanel(
                 tabPanel("Output 1"),
                 tabPanel("Output 2")
                 ) # close tabset panel
               ),# close visual output column

        column(2,
               wellPanel(
                 h3("Topline Outputs")
                 ) # close well panel
               ) # close text output column


      ) # close page fluidrow


    )# close fluidpage


  )# close dashboardbody

  # build dashboard elements --------
  dashboardPage(
    skin = "black",
    header,
    dashboardSidebar(disable = TRUE),
    body
  )

})
```

Column widths must add to 12

Closure comments are helpful!

Assemble the page here

```r
# bare bones server example
# 2022-10-04

# load -------

library(tidyverse)
library(shiny)
library(leaflet)
library(scales)
library(viridis)
library(plotly)
library(sf)

shinyServer(function(input, output, session){


})
```

Start with an empty server to build your UI

A fullly-functional simulator may *look* intimidating, but it follows the same structure as our bare-bones example, with logic added

```
# define header ---------
header <- dashboardHeader(
  disable = FALSE,
  titleWidth = 350,
  # Change this to fit your title for the Shiny App
  title = "My Shiny App Title"

)

# define body -------------
body <- dashboardBody(
  fluidPage(
    # other themes available here: https://rstudio.github.io/shinythemes/
    theme = "yeti",
    fluidRow(

      # left panel -----------------

      column(2, # this number defines the width of your column,
                # all of your columns within a fluid row should add up to 12
                # to fit the whole space - this is a standard website thing
               wellPanel(
                 id = "formula_inputs",
                 h3("Formula Inputs"), # h3 defines the font size and style,
                                       # the smaller the number, the bigger the
                                       # font (h3 is bigger than h4)
                 h4("Base Funding"),
                 fluidRow(
                   column(12,
                          numericInput("base", "Per-Pupil Base:", step = 1, value = 6860)
                          )
                 ),
                 # set a numeric input value with id = 'base'
                 # means you can call this in server.r with input$base

                 # create hover-over help text when a user hovers over the
                 # input for "base", which is defined above
                 bsTooltip("base", "The base cost to educate a student with no special needs. U
                 # another standard web development thing - this means break, aka create one line of b
                 br(),
```

```
# R Class Shiny Simulator Template
# 2022-10-04

# load ----------
library(shiny)
library(leaflet)
library(scales)
library(viridis)
library(plotly)
library(sf)
library(tidyverse)


# Read in clean simulator data from your project's data folder
# this data should be completely processed, don't do any processing in
# server.R - it will slow down your simulator

app_data <- read_rds("data/simulator_data.rds")
dist_shp <- read_sf("data/sim_dist.shp")

# define server logic ----------
shinyServer(function(input, output, session) {

  # state formula current + dynamic calculations -----------
  sim_data <- reactive({

    app_data |>
      # this is the code for the dynamic base, weights, and direct funding.
      # The $ makes these dynamic so the
      # user can change the base, weights, etc.
      # create new cols for dynamic formula vars ---------
      mutate(base_amount = input$base,
             # weights for FRPL, sped, ELL, sparsity, and concentrated poverty
             frpl_weight = input$frpl_weight,

             sped_opt1_weight = input$sped_opt1_weight,
             sped_opt2_weight = input$sped_opt2_weight,
             sped_opt3_weight = input$sped_opt3_weight,
```

The scripts used to generate synthetic data for our example simulator dashboard may be helpful as you clean your state's data

```r
# load ---------

set.seed(36)

library(tidyverse)
library(edbuildr)

dist_raw <- masterpull(data_year = "2019", data_type = "geo")

de_raw <- dist_raw |>
  filter(State == "Delaware") |>
  rename_with(tolower)

de_clean <- de_raw |>
  mutate(frpl_pct = case_when(stpovrate > .12 ~ stpovrate * 3,
                              stpovrate > .1 ~ stpovrate * 2,
                              TRUE ~ stpovrate),
         frpl_adm = enroll * frpl_pct) |>
  mutate(sped_opt1_adm = rnorm(16, mean = .12, sd = .04) * enroll,
         sped_opt2_adm = rnorm(16, mean = .06, sd = .02) * enroll,
         sped_opt3_adm = rnorm(16, mean = .03, sd = .01) * enroll) |>
  mutate(el_adm = abs(rnorm(16, mean = .05, sd = .025)) * enroll) |>
  rename(base_adm = enroll,
         district = name) |>
  select(ncesid, state_id, district, frpl_pct,
         base_adm, frpl_adm,
         sped_opt1_adm, sped_opt2_adm, sped_opt3_adm,
         el_adm,
         student_per_sq_mile,mhi, mpv)

write_rds(de_clean, "data/simulator_data.rds")
```

```r
# load ------------
library(tidyverse)
library(sf)
library(edbuildmapr)

raw_sd_shp <- sd_shapepull()

de_shp <- raw_sd_shp |>
  filter(State == "Delaware") |>
  rename_with(tolower) |>
  select(geoid, geometry) |>
  st_transform("WGS84")

# write ------

write_sf(de_shp, "data/sim_dist.shp")
```

Agenda

Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

Build up your UI panel one row at at time, adding in section headers as needed

Use the Shiny cheatsheet!

```
wellPanel(
  id = "formula_inputs",
  h3("Formula Inputs"), # h3 defines the font si    and style,
                        # the smaller t    number, the bigger the
                        # fon   (h3 is bigger than h4)
  h4("Base Funding"),
  fluidRow(
    column(12,
           numericInput("base", "Per-Pupil Base:", step = 1, value = 6860)
           )
  ),
  # set a numeric input value with id = 'base'
  # means you can call this in server.r with input$base

  # create hover-over help text when a user hovers over the
  # input for "base", which is defined above
  bsTooltip("base", "The base cost to educate a student with no special needs. Users can adjust this amount.",
            placement = "bottom"),
  # another standard web development thing - this means bl
  # aka create one line of blank space
  br(),
```

Use headers to define sections of inputs

Each new input needs its own row, then a column within that row

Inputs will have an id, a label, and a set value. Other parameters will vary by input type

Add a tooltip using the input id

Once you're done with the section, add a break

Tidy up the UI with some conditional formatting to hide elements when you're not adjusting them

Radio buttons are great inputs for binary conditionals

```
fluidRow(
  column(5, h4("Student Weights"))
),
fluidRow(
  column(12,
         radioButtons(inputId = 'frpl_adm', label = 'FRPL weight',
                      choices = c('Show', 'Hide'), inline = TRUE, selected = 'Hide')
  )
),
# create a conditional set of UI that only appears when a
# certain action is taken by the user
conditionalPanel(
  # input.____ refers to a variable generated in the UI scri
  # in this case, so if the input frpl_adm is "Show", do this
  condition = "input.frpl_adm == 'Show'",
  fluidRow(
    column(6,
           numericInput("frpl_weight", label="Free or Reduced-Price Lunch", min=0, max=1.5, step=.01, value=0.25)
    ),
    bsTooltip("frpl_weight", "The percent of additional funding, relative to the base, that each qualifying student recei
              placement = "bottom")
  )
),
```

The conditionalPanel will only appear if a condition is met

Once the condition is defined, you can add an input as usual

Agenda

Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

Agenda

Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

In your state teams, start working to build out a vision for your simulator's UI elements

- Identify the elements you want to include in your simulator
  - Match those elements to input types (use the Shiny cheatsheet)
  - Decide how you want to organize your inputs (groups, conditionals)
  - Start coding!
- Next, decide on the kinds of visual outputs you'll want
  - Organize those outputs into tabs
  - Start coding!
- Finally, decide on your text outputs
  - Organize those outputs
  - Start coding!

Agenda

Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

Building your back-end logic will be challenging, but taking it one step at a time will save you hours of headaches

```
# Read in clean simulator data from your project's data folder
# this data should be completely processed, don't do any processing in
# server.R - it will slow down your simulator

app_data <- read_rds("data/simulator_data.rds")
dist_shp <- read_sf("data/sim_dist.shp")

# define server logic ----------
shinyServer(function(input, output, session) {

  # state formula current + dynamic calculations --------
  sim_data <- reactive({

    app_data |>
      # this is the code for the dynamic base, weights, and direct funding.
      # The $ makes these dynamic so the
      # user can change the base, weights, etc.
      # create new cols for dynamic formula vars ---------
      mutate(base_amount = input$base,
             # weights for FRPL, sped, ELL, sparsity, and concentrated pove
             frpl_weight = input$frpl_weight,

             sped_opt1_weight = input$sped_opt1_weight,
             sped_opt2_weight = input$sped_opt2_weight,
             sped_opt3_weight = input$sped_opt3_weight,

             el_weight = input$el_weight,

             sparsity_limit = input$sparsity_limit,
             conc_pov_min = input$conc_pov_min,

             sparsity_weight = input$sparsity_weight,
             conc_pov_weight = input$conc_pov_weight,

             # dynamic adm counts -------------

             # set sparsity adm to base adm if s p
             # else set sparsity adm to zero if s
             sparsity_adm = ifelse(student_per_sq_
                                   base_adm,
                                   0),
```

Make sure you're starting with data you've already cleaned!

The core of your simulator logic is building a reactive dataframe that takes your raw app data and applies a `mutate` that includes all of the steps for your current formula and dynamic formula alternative

Start by creating columns that will store your dynamic input values

Bellwether

The actual core of your simulator logic will probably involve simple math — breaking things into discrete steps will make your coding experience easier

```
# dynamic formula code ---------------
# this code calculates the new funding amounts based on the dynamic
# base and weight amounts
new_base_funding = base_amount * base_adm,
# New weight amounts
new_frpl_total = base_amount * (frpl_adm * frpl_weight),
new_sped_opt1_total = base_amount * (sped_opt1_adm * sped_opt1_weight),
new_sped_opt2_total = base_amount * (sped_opt2_adm * sped_opt2_weight),
new_sped_opt3_total = base_amount * (sped_opt3_adm * sped_opt3_weight),
new_el_total = base_amount * (el_adm * el_weight),

new_sparsity_total = base_amount * (sparsity_adm * sparsity_weight),
new_conc_pov_total = base_amount * (conc_pov_adm * conc_pov_weight),

# |- dynamic formula totals ----------
# new state funding total
new_state_funding_total = new_base_funding + new_frpl_total +
  new_sped_opt1_total + new_sped_opt2_total + new_sped_opt3_total +
  new_el_total +
  new_sparsity_total + new_conc_pov_total,

new_state_pp = new_state_funding_total / base_adm,

# New weights funding
new_weights_total = new_frpl_total +
  new_sped_opt1_total + new_sped_opt2_total + new_sped_opt3_total +
  new_el_total +
  new_sparsity_total + new_conc_pov_total,

# current vs dynamic district differences -------
weights_diff = new_weights_total - current_weights_total,
state_total_diff = new_state_funding_total - current_state_funding_total,
state_total_pp = new_state_pp - current_state_pp)
```

Your dynamic funding model will use the input columns you created earlier

Create columns to illustrate differences between dynamic and current funding

Building visual elements for your simulator should be pretty familiar by this point – it's all ggplot, plotly, and leaflet (with some tweaks)

```
# plot 1 output ----------------
  output$plot1 <- renderPlotly({
    ggplotly(
      ggplot(sim_data(),
                    aes(x = mpv,
                        y = new_state_pp,
                        size = base_adm,
                        color = frpl_pct,
                        text = paste0("District: ", district, "<br>",
                                      "Property Wealth PP: ", dollar(mpv, accuracy = 1), "<br>",
                                      "Dyanmic State PP: ", dollar(new_state_pp, accuracy = 1), "<br>",
                                      "Base ADM: ", comma(base_adm), "<br>",
                                      "FRPL %: ", percent(frpl_pct, accuracy = .1)))) + # tooltip definition
               geom_point() +
               scale_x_continuous(labels = dollar_format()) +
               scale_y_continuous(labels = dollar_format()) +
               scale_size_area(labels = comma_format()) +
               scale_color_viridis(labels = percent_format(accuracy = 1)) +
               labs(x = "Property Wealth Per-Pupil",
                    y = "Dynamic State Aid Per-Pupil",
                    color = "FRPL %",
                    size = "Base ADM") +
               theme_bw(),
      tooltip = "text")
  })
```

Visual outputs all need to be wrapped in a `renderXXXX` function and stored as an output

Text output construction is probably the most straightforward portion of server-side logic construction

```r
# line item totals ------------
## define the outputs for the total state budget and the balance to show
output$budget <- reactive({
  dollar(sum(sim_data()$current_state_funding_total, na.rm = TRUE))
})

output$balance <- reactive({
  dollar(sum(sim_data()$new_state_funding_total) - sum(sim_data()$current_state_funding_total, na.rm = TRUE))
})
```

Text output creation is as easy as creating a single reactive value that is based on your formula df

```r
# data for download button ---------

df_for_dl <- reactive({
  # this is really helpful for debugging since you can change which df()
  # will be passed through to the dl function below - just make sure only one
  # of these dfs is un-commented!
  sim_data()
  # state_summary()
  # dist_summary()
})

# this function will allow you to download a .csv of your data
output$download_data <- downloadHandler(

  filename = function() {
    # this names the csv file with today's date
    paste('simulator-output-', Sys.Date(), '.csv', sep='')
  },
  content = function(file) {
    write_csv(df_for_dl(), file)
  }

) # close downloadHandler
```

Use the download function to help check your math and debug issues with your formula code!

Agenda

Agenda	Time
Homework review	10 Minutes
Intro to shiny dashboards	20 minutes
Building the UI panel	25 minutes
Break	10 minutes
State group UI development	20 minutes
Adding in the back-end logic	30 minutes
Closing / Homework	5 minutes

Homework for next class

- Start building out your state school finance simulator!
- Focus on getting the UI elements locked in *first*
- Once your UI is solid, start adding in elements to the server logic
- **Send a state team update to Alex and Krista by noon on Monday, October 10!**