AE 23: Building a climate risk dashboard - server (output)

Suggested answers

APPLICATION EXERCISE ANSWERS

MODIFIED
April 24, 2025

Communicating climate risk

FEMA has asked us to build an improved dashboard that visualizes the risk of climate change in the United States.

Based on your submitted designs and our skill level, we will work to implement this Shiny dashboard.

Construct the server (output)

National map

- [x] Create the national map and store it in `output$national_map`. Use the provided templated code to get started.
- [x] Add a `message()` to print an informative message in the console whenever `output$national_map` is rendered. Using `message()` and `warning()` functions makes it much easier to troubleshoot the app.
- [x] Render the dashboard and test the national map. Does it work as expected?

Starter code to generate the national map

Fill in the `TODO`s and correctly integrate into the server code chunk.

```
# visualize the risks nationally
ggplot(data = climate_sf) +
  # layer for each county's risk
  geom_sf(mapping = aes(fill = TODO)) +
  # layer for state boundaries to better locate regions
  geom_sf(data = state_sf, fill = NA, color = "white") +
  # appropriate color palette
  scale_fill_discrete_diverging(rev = TRUE) +
  # don't label the legend
  labs(
    fill = NULL
  ) +
```

```
  # map theme
  theme_map(base_size = 18) +
  # position the legend on the right
  theme(legend.position = "right")
```

Tip

To access input values which are character strings and treat them as column names, use the **bang-bang operator**. For example, if `input$var_name` contained the name of a column and you wanted to use it in a {ggplot2} visualization, you could write your code like:

```
output$myplot <- renderPlot({
  ggplot(data = df, mapping = aes(x = !!input$var_name)) +
    geom_histogram()
})
```

County map

- [x] Create the highlighted county map and store it in `output$county_map`. Use the provided templated code to get started.
- [x] Add a `message()` to print an informative message in the console whenever `output$county_map` is rendered. Using `message()` and `warning()` functions makes it much easier to troubleshoot the app.

Starter code to generate the national map

Fill in the `TODO`s and correctly integrate into the server code chunk.

```
# get selected county's state
county_state <- climate_sf |>
  filter(county == TODO) |>
  pull(STATE_NAME)

climate_sf |>
  # filter for counties in specific state
  filter(STATE_NAME == county_state) |>
  # variable to highlight selected county
  mutate(selected = county == TODO) |>
  # draw the map
  ggplot() +
  geom_sf(mapping = aes(fill = selected, color = selected)) +
  scale_fill_manual(values = c(NA, "orange")) +
  scale_color_manual(values = c("white", "orange")) +
  theme_map() +
  theme(legend.position = "none")
```

Validating inputs

What happens if there is no county provided in the input? Cryptic red error messages in the UI can be disconcerting to users. Instead, we can use `validate()` and `need()` to shunt the error messages to the console and print a human-readable message to the user.

For example, say the output relies on `input$num` which must not be empty. We can use

```
need(input$num, "Provide a number")
```

to check if `input$num` is provided. If the condition fails (i.e. `input$num` is blank or missing), then `need()` returns the specified character string. We can incorporate these requirements into the output using `validate()`.

```
output$myplot <- renderPlot({
  validate(
    need(input$num, "Provide a number")
  )

  # remaining code...
})
```

`validate()` takes any number of arguments depending on how many conditions must be checked. If any of them fail, the app triggers an error which is reported in the console and displays the `need()`-provided message to the user in the UI.

- [x] Add an appropriate validation check to `output$county_map` that ensures the user has provided a specific county.

County risk scores

`selected_county`

- [x] We will need to repeatedly query the `climate_risk` data frame for specific information on selected counties. Rather than repeat this process, instead create `selected_county` which is a filtered copy of the `climate_risk` data frame for the specified county.
- [x] Add an appropriate validation check to `selected_county` that ensures the user has provided a specific county.

County stats for value boxes

Use the filtered data frame to generate the text strings for each of the value boxes.

- [x] Overall risk
- [x] Expected annual loss
- [x] Social vulnerability

- [x] County resilience

Starter code to generate the national map

```
# overall risk
output$county_risk <- renderText({
  # get selected county's overall risk score
  val <- selected_county() |>
    pull(national_risk_index_score_composite)

  # format using scales function
  label_number(accuracy = 1)(val)
})
```

Individual hazard percentiles

- [x] Generate the county hazards dot plot and store in `output$county_hazards`.
- [x] Incorporate appropriate validation checks for the inputs.

Starter code to generate the national map

Fill in the `TODO`s and correctly integrate into the server code chunk.

```
# get selected county's hazard percentiles
selected_county_hazards <- climate_risk |>
  filter(county == TODO)

# plot hazard percentiles
selected_county_hazards |>
  # reshape to long format for visualizing
  pivot_longer(
    cols = contains("hazard"),
    names_to = "hazard",
    values_to = "percentile"
  ) |>
  # only visualize selected hazard types
  filter(hazard %in% TODO) |>
  # order alphabetically
  mutate(hazard = str_remove(hazard, "_hazard_type_risk_index_score") |>
           make_clean_names(case = "title") |>
           fct_rev()) |>
  ggplot(mapping = aes(y = hazard)) +
  # all hazards range between 0 and 100
  geom_linerange(mapping = aes(xmin = 0, xmax = 100)) +
  # draw specific county
  geom_point(
    mapping = aes(x = percentile, color = percentile),
    size = 4
```

```
  ) +
  # optimized color palette
  scale_color_continuous_diverging(mid = 50, rev = TRUE, guide = "none") +
  # appropriate labels
  labs(
    x = "Percentile",
    y = NULL
  ) +
  # clean up the theme
  theme_minimal(base_size = 18) +
  theme(
    panel.grid.minor = element_blank(),
    panel.grid.major.y = element_blank()
  )
```

Suggested code

```
library(tidyverse)
library(scales)
library(shiny)
library(sf)
library(janitor)
library(ggthemes)
library(colorspace)
library(bslib)
library(bsicons)
library(gt)

# Import data ------------------------------------------------------------------
# climate risk
climate_risk <- read_rds(file = "data/climate-risk.rds")

# import state and county boundaries
state_sf <- st_read(dsn = "data/states.geojson")
county_sf <- st_read(dsn = "data/counties.geojson")

# combine climate risk with county_sf
climate_sf <- left_join(
  x = county_sf,
  y = climate_risk,
  by = join_by(GEOID == state_county_fips_code)
) |>
  as_tibble() |>
  st_as_sf()
```

```
# get county names
county_names <- climate_sf |>
  arrange(STATEFP) |>
  pull(county)

# define hazard types
hazard_types <- climate_risk |>
  select(contains("hazard")) |>
  colnames()

# create human-readable labels
hazard_types_labels <- hazard_types |>
  str_remove(pattern = "_hazard_type_risk_index_score") |>
  make_clean_names(case = "title")

# create a named character vector for the input
names(hazard_types) <- hazard_types_labels

# Define UI ----------------------------------------------------------------
ui <- page_navbar(
  title = "National Risk Index Counties",
  theme = bs_theme(version = 5, preset = "minty"),

  # National Risk Index page
  nav_panel(
    title = "National Risk Index",
    layout_sidebar(
      sidebar = sidebar(
        # select between the four risk ratings
        varSelectInput(
          inputId = "risk_var",
          label = "Risk index",
          # select specific columns of data to populate select options
          data = climate_risk |>
            select(`National Risk Index`, `Expected Annual Loss`, `Social Vulnerability`,
          `Community Resilience`)
        )
      ),
      # Main content
      card(
        card_header("National Risk Map"),
        plotOutput(outputId = "national_map")
      )
```

```
    )
  ),

  # County Details page
  nav_panel(
    title = "County Details",
    layout_sidebar(
      sidebar = sidebar(
        # extract county/state labels as character vector
        selectizeInput(
          inputId = "county",
          label = "Selected county",
          choices = county_names,
          selected = NULL,
          # custom selectize.js options
          options = list(
            # placeholder text
            placeholder = "Select a county",
            # limit to one county at a time
            maxItems = 1
          )
        ),

        # identify columns with hazard risks and extract column names
        checkboxGroupInput(
          inputId = "hazard_types",
          label = "Hazard types",
          # all possible choices
          choices = hazard_types,
          # initialize plot with all individual hazards
          selected = hazard_types
        )
      ),
      # Main content
      layout_column_wrap(
        width = "400px", # This width makes two columns when screen is > 800px, one column
          when narrower
        style = htmltools::css(gap = "10px", margin_bottom = "10px"),
        # Row 1 - Maps side by side on wider screens
        card(
          card_header("County Map"),
          plotOutput(outputId = "county_map")
        ),
        card(
```

```
          card_header("County Hazards"),
          plotOutput(outputId = "county_hazards")
        )
      ),
      layout_column_wrap(
        width = 1/4,
        height = "auto",
        style = htmltools::css(gap = "10px"),
        # Row 2 - Value boxes
        value_box(
          title = "Overall risk score",
          value = textOutput("county_risk"),
          showcase = bs_icon("radioactive")
        ),
        value_box(
          title = "Expected annual loss",
          value = textOutput("county_loss"),
          showcase = bs_icon("trash")
        ),
        value_box(
          title = "Social vulnerability",
          value = textOutput("county_vulnerability"),
          showcase = bs_icon("cone-striped")
        ),
        value_box(
          title = "Community resilience",
          value = textOutput("county_resilience"),
          showcase = bs_icon("emoji-sunglasses")
        )
      )
    )
  ),

  # Data page
  nav_panel(
    title = "Data",
    card(
      card_header("National Risk Index Data"),
      climate_risk |>
        gt() |>
        opt_interactive()
    )
  )
)
```

```
# Server function for the National Risk Index Counties Shiny app
server <- function(input, output) {

  # national map
  output$national_map <- renderPlot({
    # print message to console for logging
    message("Rendering national map")

    # visualize the risks nationally
    ggplot(data = climate_sf) +
      # layer for each county's risk
      geom_sf(mapping = aes(fill = !!input$risk_var)) +
      # layer for state boundaries to better locate regions
      geom_sf(data = state_sf, fill = NA, color = "white") +
      # appropriate color palette
      scale_fill_discrete_diverging(rev = TRUE) +
      # don't label the legend
      labs(
        fill = NULL
      ) +
      # map theme
      theme_map(base_size = 18) +
      # position the legend on the right
      theme(legend.position = "right")
  })

  # county map
  output$county_map <- renderPlot({
    # print message to console for logging
    message("Rendering county map")

    # check that input$county is valid to avoid error messages in app
    validate(
      need(input$county, "Select a county")
    )

    # get selected county's state
    county_state <- climate_sf |>
      filter(county == input$county) |>
      pull(STATE_NAME)

    climate_sf |>
      # filter for counties in specific state
```

```
    filter(STATE_NAME == county_state) |>
    # variable to highlight selected county
    mutate(selected = county == input$county) |>
    # draw the map
    ggplot() +
    geom_sf(mapping = aes(fill = selected, color = selected)) +
    scale_fill_manual(values = c(NA, "orange")) +
    scale_color_manual(values = c("white", "orange")) +
    theme_map() +
    theme(legend.position = "none")
})

# filtered county observation
selected_county <- reactive({
  # check that input$county is valid to avoid error messages in app
  validate(
    need(input$county, "Select a county")
  )

  climate_risk |>
    filter(county == input$county)
})

###### county stats for value boxes
# overall risk
output$county_risk <- renderText({
  # get selected county's overall risk score
  val <- selected_county() |>
    pull(national_risk_index_score_composite)

  # format using scales function
  label_number(accuracy = 1)(val)
})

# expected loss
output$county_loss <- renderText({
  # get selected county's overall risk score
  val <- selected_county() |>
    pull(expected_annual_loss_score_composite)

  # format using scales function
  label_number(accuracy = 1)(val)
})
```

```
  # social vulnerability
  output$county_vulnerability <- renderText({
    # get selected county's overall risk score
    val <- selected_county() |>
      pull(social_vulnerability_score)

    # format using scales function
    label_number(accuracy = 1)(val)
  })

  # community resilience
  output$county_resilience <- renderText({
    # get selected county's overall risk score
    val <- selected_county() |>
      pull(community_resilience_score)

    # format using scales function
    label_number(accuracy = 1)(val)
  })

  # individual hazard percentiles
  output$county_hazards <- renderPlot({
    # print message to console for logging
    message("Rendering county hazards dot plot")

    # check that input$county and input$hazard_types is valid to avoid error messages in app
    validate(
      need(input$county, "Select a county"),
      need(input$hazard_types, "Select hazard types")
    )

    # get selected county's hazard percentiles
    selected_county_hazards <- climate_risk |>
      filter(county == input$county)

    # plot hazard percentiles
    selected_county_hazards |>
      # reshape to long format for visualizing
      pivot_longer(
        cols = contains("hazard"),
        names_to = "hazard",
        values_to = "percentile"
      ) |>
      # only visualize selected hazard types
```

```
      filter(hazard %in% input$hazard_types) |>
      # order alphabetically
      mutate(hazard = str_remove(hazard, "_hazard_type_risk_index_score") |>
               make_clean_names(case = "title") |>
               fct_rev()) |>
      ggplot(mapping = aes(y = hazard)) +
      # all hazards range between 0 and 100
      geom_linerange(mapping = aes(xmin = 0, xmax = 100)) +
      # draw specific county
      geom_point(
        mapping = aes(x = percentile, color = percentile),
        size = 4
      ) +
      # optimized color palette
      scale_color_continuous_diverging(mid = 50, rev = TRUE, guide = "none") +
      # appropriate labels
      labs(
        x = "Percentile",
        y = NULL
      ) +
      # clean up the theme
      theme_minimal(base_size = 18) +
      theme(
        panel.grid.minor = element_blank(),
        panel.grid.major.y = element_blank()
      )
  })
}

# Run the app
shinyApp(ui = ui, server = server)
```