Session 3: Data Visualization

Dan Killian and Ted Papalexopoulos

15.S60 COS 2021

14 January 2021

Email tpapalex@mit.edu if you want access to a Google doc of these slides.

Agenda

- Recap of Tidyverse
- Grammar of Graphics (ggplot)
- Intro to R Shiny

What is the Grammar of Graphics?

- A unifying approach to graphics...
  - Created by Leland Wilkinson in The Grammar of Graphics
  - Implemented as `ggplot` in R by Hadley Wickham
  - Part of the `tidyverse`

What is the Grammar of Graphics?

- Why *grammar?*
  - A set of guidelines for combining **elements** to make valid **composites**
  - In English, combine *nouns* (“dog”) and *verbs* (“run”) to make *sentences* (“the dog runs”)
  - In visualization, combine *???* to make *plots*

The Elements of a Plot

Every `ggplot` consists of four main elements:

- Data the data we want to plot
- Aesthetics mapping of data columns to “dimensions” (e.g. x, y, color, shape)
- Geometry the specific visualization shape
- Theme fine-grained appearance

The Elements of a Plot

Every `ggplot` consists of four main elements:

- Data the data we want to plot
  - AirBnB listings dataframe
- Aesthetics mapping of data columns to “dimensions” (e.g. x, y, color, shape)
  - “price” on the x-axis, “rating” on the y-axis
- Geometry the specific visualization shape
  - Line plot, scatter plot, bar chart...
- Theme fine-grained appearance
  - E.g. axis labels, text formatting, background

The Elements of a Plot

Every `ggplot` consists of four main elements:

- Data the data we want to plot
  - AirBnB listings dataframe
- Aesthetics mapping of data columns to “dimensions” (e.g. x, y, color, shape)
  - “price” on the x-axis, “rating” on the y-axis
- Geometry the specific visualization shape
  - Line plot, scatter plot, bar chart...
- Theme fine-grained appearance
  - E.g. axis labels, text formatting, background

```
data %>% ggplot(aes) + geom + theme
```

Let’s jump in...

What is Shiny?

- Want to build a GUI but only know R? Use Shiny!
- Interactively explore data:
  - Change model parameters
  - Filter datasets
  - Add extra “dimensions” to your visualizations
- Probably the easiest way to impress stakeholders
  - Users can interact with your data and analysis
  - You can host standalone apps on a webpage or build dashboards within your code.

Anatomy of Shiny App

```
shinyApp(ui = ui, server = server)
```

- **UI**: defines webpage layout, buttons, plots, etc.
- **Server**: Processes inputs into outputs (i.e., everything else)

Anatomy of a Shiny App: **UI**

A UI contains **Layouts**, **Inputs** and **Outputs**

```
ui <- fluidPage(
    verticalLayout(
        sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),
        plotOutput("distPlot")
    )
)
```

Anatomy of a Shiny App: **UI**

Layouts define how objects are placed on the webpage

```
ui <- fluidPage(
    verticalLayout(
        sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),
        plotOutput("distPlot")
    )
)
```

Anatomy of a Shiny App: **UI**

Inputs define controls for the user (e.g., sliderInput, dateInput, fileInput)

```
ui <- fluidPage(
    verticalLayout(
        sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),
        plotOutput("distPlot")
    )
)
```

Anatomy of a Shiny App: **UI**

Outputs define things to display

```
ui <- fluidPage(
    verticalLayout(
        sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),
        plotOutput("distPlot")
    )
)
```

Anatomy of a Shiny App: **UI**

Inputs and Outputs have **IDs** that the server uses to access their values

```
ui <- fluidPage(
    verticalLayout(
        sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),
        plotOutput("distPlot")
    )
)
```

Anatomy of a Shiny App: **Server**

The **Server** is a function. It takes a list of **inputs**, processes them using **reactives**, and assigns the results to a list of **outputs**

```
server <- function(input, output) {

    x    <- faithful[, 2]

    bins <- reactive({

        seq(min(x), max(x), length.out = input$bins + 1)

    })

    output$distPlot <- renderPlot({

        hist(x, breaks = bins())

    })

}
```

Anatomy of a Shiny App: **Server**

Inputs and outputs from the UI are accessible by their **ID.**

```
server <- function(input, output) {

    x    <- faithful[, 2]

    bins <- reactive({

        seq(min(x), max(x), length.out = input$bins + 1)

    })

    output$distPlot <- renderPlot({

        hist(x, breaks = bins())

    })

}
```

Anatomy of a Shiny App: **Server**

Reactives are functions in the server that are executed whenever their inputs change (more on this later). Objects that depend on the input **must be wrapped in a reactive**.

```
server <- function(input, output) {

    x    <- faithful[, 2]

    bins <- reactive({

        seq(min(x), max(x), length.out = input$bins + 1)

    })

    output$distPlot <- renderPlot({

        hist(x, breaks = bins())

    })

}
```

Anatomy of a Shiny App: **Server**

Objects that don’t depend on the input don’t have to be inside reactives.

```
server <- function(input, output) {

    x    <- faithful[, 2]

    bins <- reactive({

        seq(min(x), max(x), length.out = input$bins + 1)

    })

    output$distPlot <- renderPlot({

        hist(x, breaks = bins())

    })

}
```

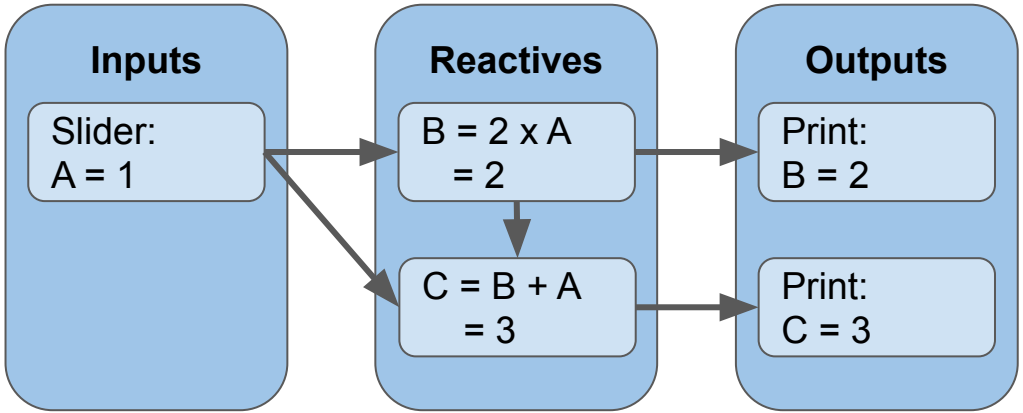
Anatomy of a Shiny App: **Server**

Reactives are typically defined using `reactive({ … })`. Reactives that generate outputs are special and correspond to the type of output: `renderPlot` corresponds to `plotOutput` in the UI, etc.

```
server <- function(input, output) {

    x    <- faithful[, 2]

    bins <- reactive({

        seq(min(x), max(x), length.out = input$bins + 1)

    })

    output$distPlot <- renderPlot({

        hist(x, breaks = bins())

    })

}
```

Reactivity

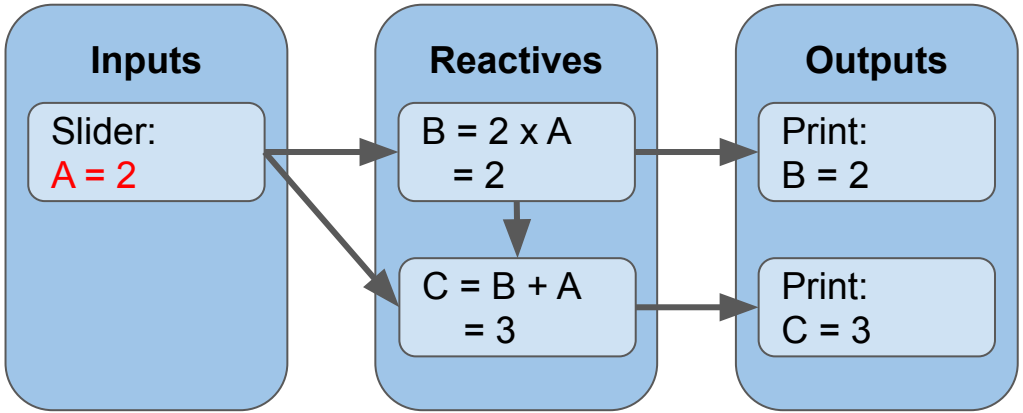
- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



Reactivity

- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

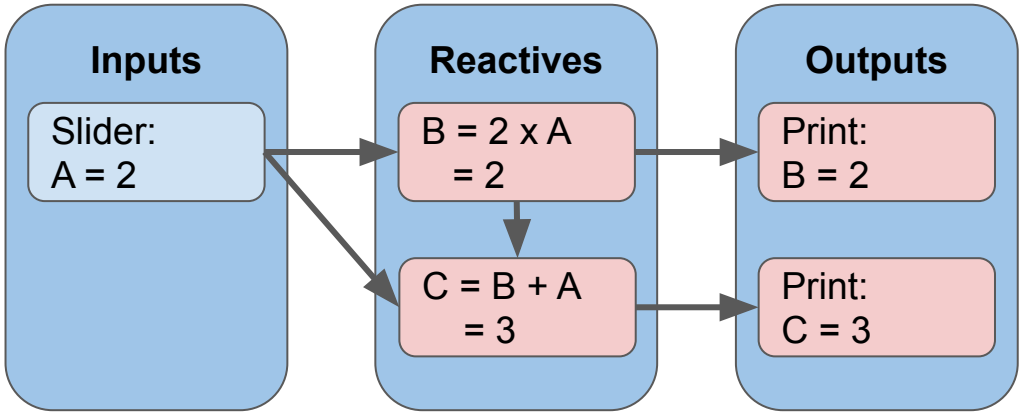
1. Change input: A -> 2



Reactivity

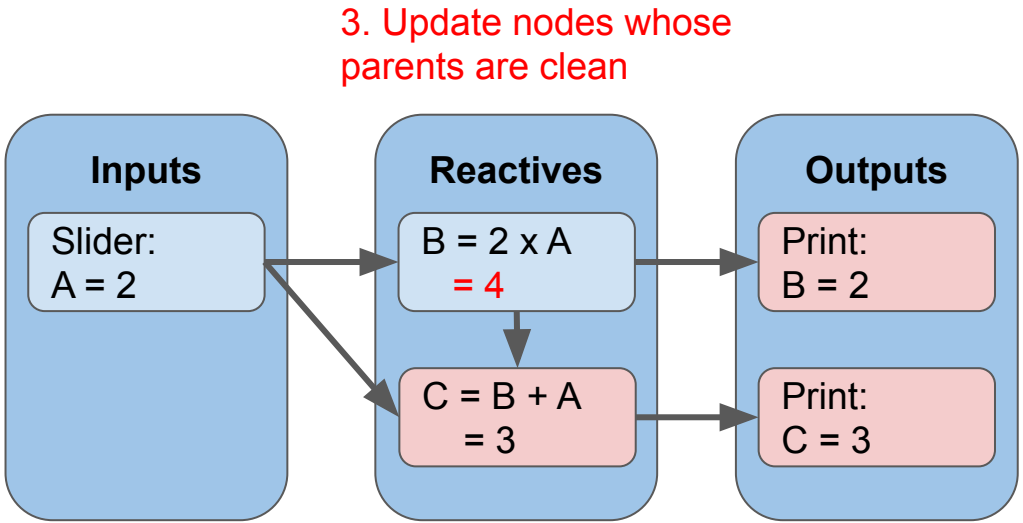
- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

2. Mark all downstream nodes as “dirty”



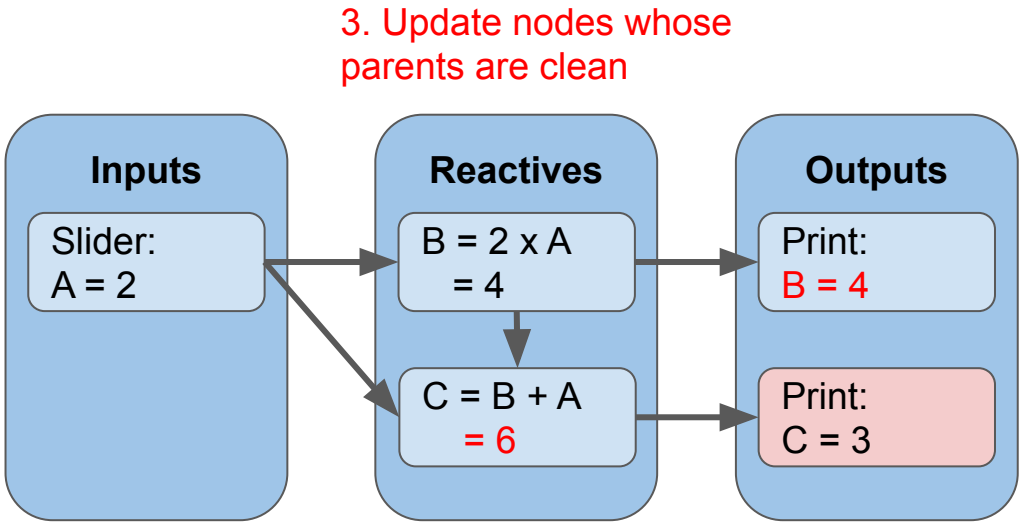
Reactivity

- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



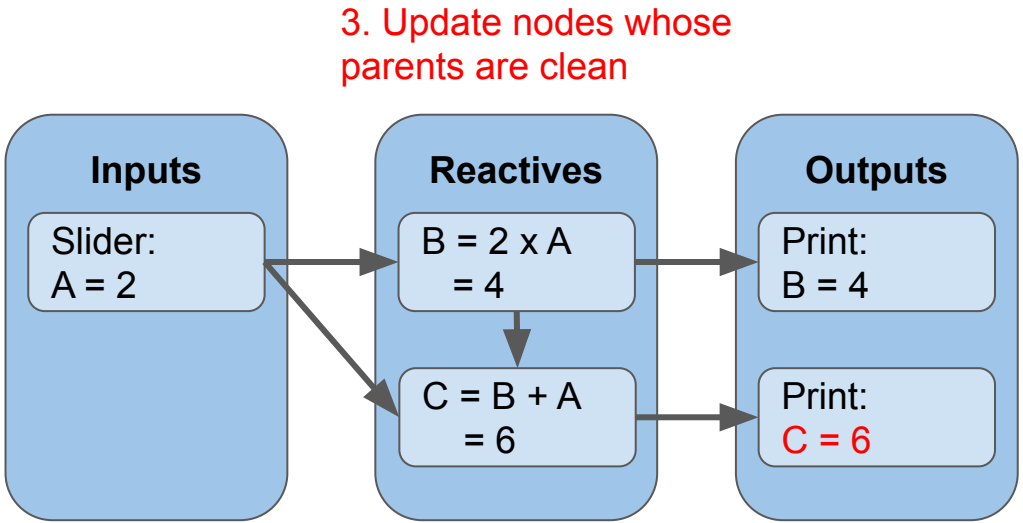
Reactivity

- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



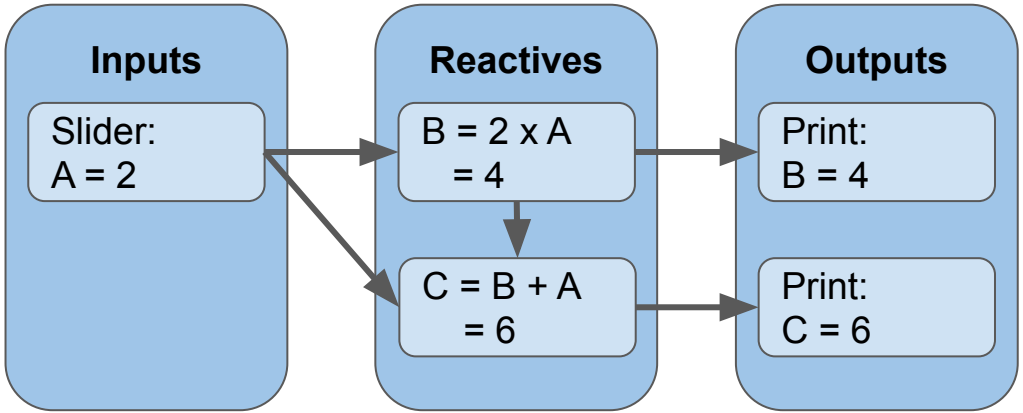
Reactivity

- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



Reactivity

- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



4. Done! Graph is clean;
wait for input

Demo: Building Airbnb’s UI

Shiny Resources

- Full tutorial on Shiny
- Detailed gallery with templates and examples
- Many wrappers for fancy JavaScript viz libraries
- Bootstrap themes
- Use the reticulate package to run Python from R/RStudio/Shiny
- Make an account on shinyapps.io to host your apps online