Applications

Of Data Science

Intro to Building Data Apps

Applications of Data Science - Class Bonus

Giora Simchoni

`gsimchoni@gmail.com and add #dsapps in subject`

Stat. and OR Department, TAU

2022-01-03

APPLICATIONS
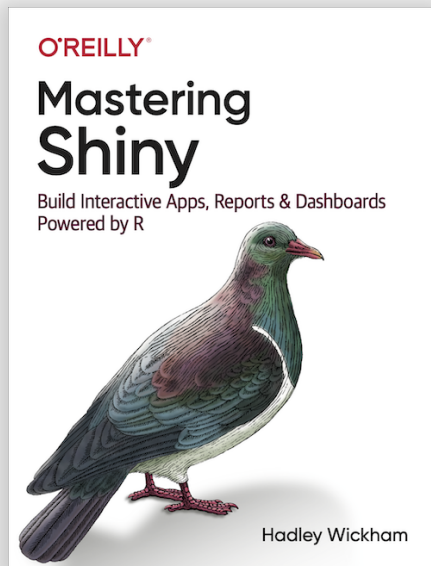
OF DATA SCIENCE

Shiny in Four Apps

Shiny

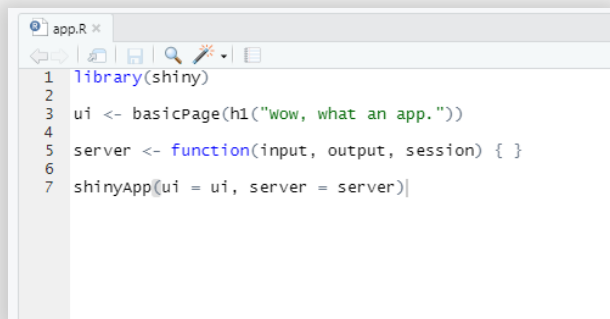Shiny is made in RStudio.

Start with the [docs](.).

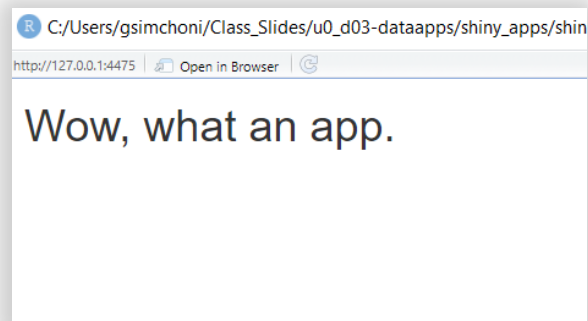Or go to Zev Ross 40 (!) apps [tutorial](.).

Or straight to God Himself:

shiny01

A single `app.R` file containing your frontend (`ui`) and backend (`server`):

shiny02

I recommend befriending the frontend (`ui`) first:

shiny03

Once it becomes too much we go modular.

Backend (`server.R`) is where R does her thing.

`observeEvent()` of slider changing to re-render a plot:

shiny04

Use `reactiveValues()` to keep the state of dynamic objects:

```
server <- function(input, output) {
  rv <- reactiveValues(
    plot = NULL,
    data = mtcars
  )

  observeEvent(input$file, {
    rv$data <- read_csv(input$file$datapath)
  })

  observeEvent(
    input$button, {
      col1 <- input$col1
      col2 <- input$col2

      rv$plot <- ggplot(rv$data %>% slice(1:input$slider1)) +
        aes_string(col1, col2) + geom_point(size=5) +
        labs(title = input$text) +
        theme_light()
    }
  )
  output$plot <- renderPlot({
    if (is.null(rv$plot)) return()
    rv$plot
  })
}
```

Use `renderUI()` for dynamic UI:

```
mainPanel(
  tabsetPanel(
    tabPanel("Tab",
             basicPage(
               column(6,
                      fileInput("file", "Choose a file:", acce
                      uiOutput("col1"),
                      uiOutput("col2")
               ),
               column(6,
                      plotOutput("plot"))
             )
```

```
output$col1 <- renderUI({
  selectInput("col1", "Select X var:", colnames(rv$data))
})

output$col2 <- renderUI({
  selectInput("col2", "Select Y var:", colnames(rv$data))
})
```

APPLICATIONS
OF DATA SCIENCE

Is that it?

Formulan

If you really want to be amazed

Visit the annual RStudio Shiny contest and the Shiny gallery.

APPLICATIONS

OF DATA SCIENCE

Dash in Four Apps

Dash

Dash is made by Plotly, other than Python it works with R and Julia.
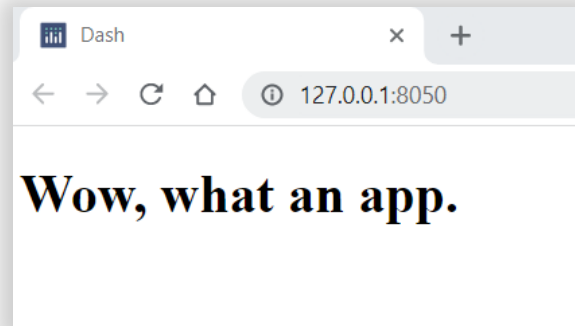
It is much "closer" to JavaScript (advantage?)

Start with the [docs](#).

Another promising option is [Voila](#) by Jupyter.

dash01

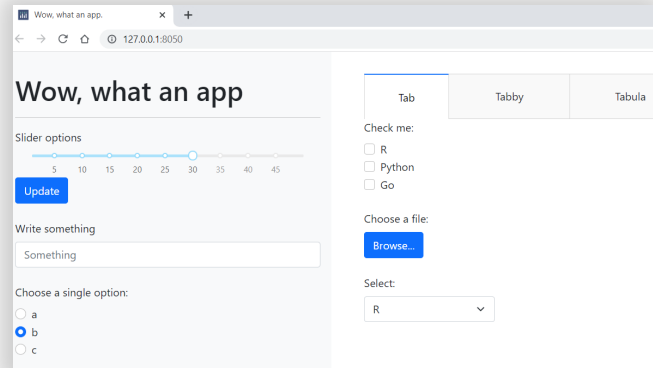A single `app.py` file containing your frontend (`layout`) and backend (`callbacks`):

```python
from dash import Dash, html

app = Dash(__name__)

app.layout = html.Div(children=[
    html.H1(children='Wow, what an app.'),
])

if __name__ == '__main__':
    app.run_server()
```

dash02

I recommend befriending the frontend (`layout`) first:

dash03

Once it becomes too much we go modular.

Backend (`callbacks.py`) is where Python does her thing.

`@app.callback()` of slider changing to re-render a plot:

dash04

There are no `reactiveValues` in Dash backend (AFAIK), but we can do multiple `Output`s/`Input`s and `State`s

```python
@app.callback(Output('plot', 'figure'),
    [Input('button', 'n_clicks'), State('slider1', 'value'), State('text', 'value'),
        State('file', 'contents'), State('col1', 'value'), State('col2', 'value')],
def update_graph(n_clicks, slider_value, title, file_content, col1, col2):
    if file_content is not None:
        df = parse_contents(file_content)
        fig = px.scatter(df.iloc[1:(slider_value + 1), :],
            x=col1, y=col2, title=title)
    else:
        fig = px.scatter(tips.iloc[1:(slider_value + 1), :],
            x=col1, y=col2, title=title)
    return fig
```

And rendering UI is very easy because every object's components are modifiable:

```python
select_option1 = html.Div([
        dbc.Label('Select X var:'),
        dbc.Select(id='col1'),
    ], style={"width": "50%"})

select_option2 = html.Div([
        dbc.Label('Select Y var:'),
        dbc.Select(id='col2'),
    ], style={"width": "50%"})
```

```python
@app.callback(Output('col1', 'options'), Output('col2', 'options'),
    Output('col1', 'value'), Output('col2', 'value'),
    [Input('file', 'contents')])
def update_dropdown(file_content):
    if file_content is not None:
        df = parse_contents(file_content)
        options = [{'label': i, 'value': i} for i in df.columns]
    else:
        options = [{'label': i, 'value': i} for i in tips.columns]
    return options, options, options[0]['value'], options[1]['value']
```

Is that it?

If you really want to be amazed

Visit the Dash [gallery](#).

Summary

Do I think you can replace the Front-end engineer at your organization? No.

But you can certainly use data apps for:

- Inside dashboards (everyone can access via company server or with Docker: Vivian)

- Personal tools (RateImagesApp, Formulan)

- Quick prototypes

- Showing people in company how data/analysis looks like and letting them playing with it

- Simulations

- Model testing