DSCI 532

Data Visualization 2

2. Building a basic Dash/Shiny app

Lecture learning goals

1. Motivate the need to use a dashboard framework
2. Understand the pros and cons with Dash and Shiny
3. Identify how a Dash/Shiny app is designed conceptually
4. Run a Dash/Shiny app on a local server
5. Create input widgets (dropdowns, sliders, etc)
6. Set up callbacks that update the app based on user input
7. Create an Altair/ggplot charts in dashboards
8. Update charts via input widgets

Required activities

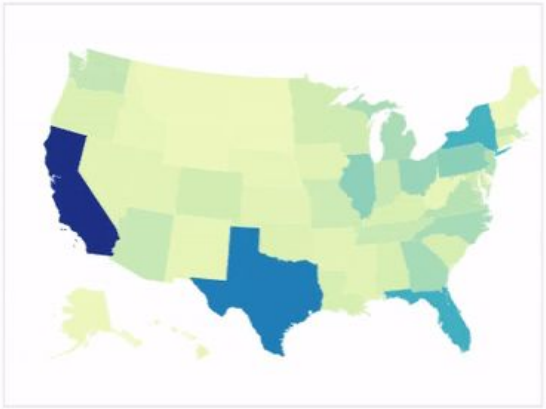
Before class:

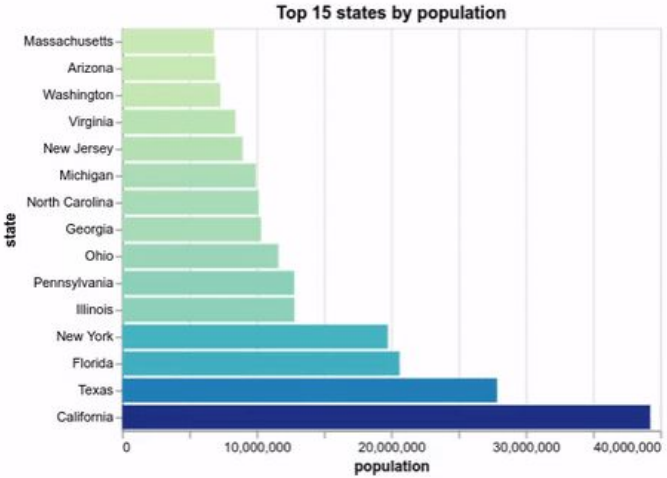
- Nothing!

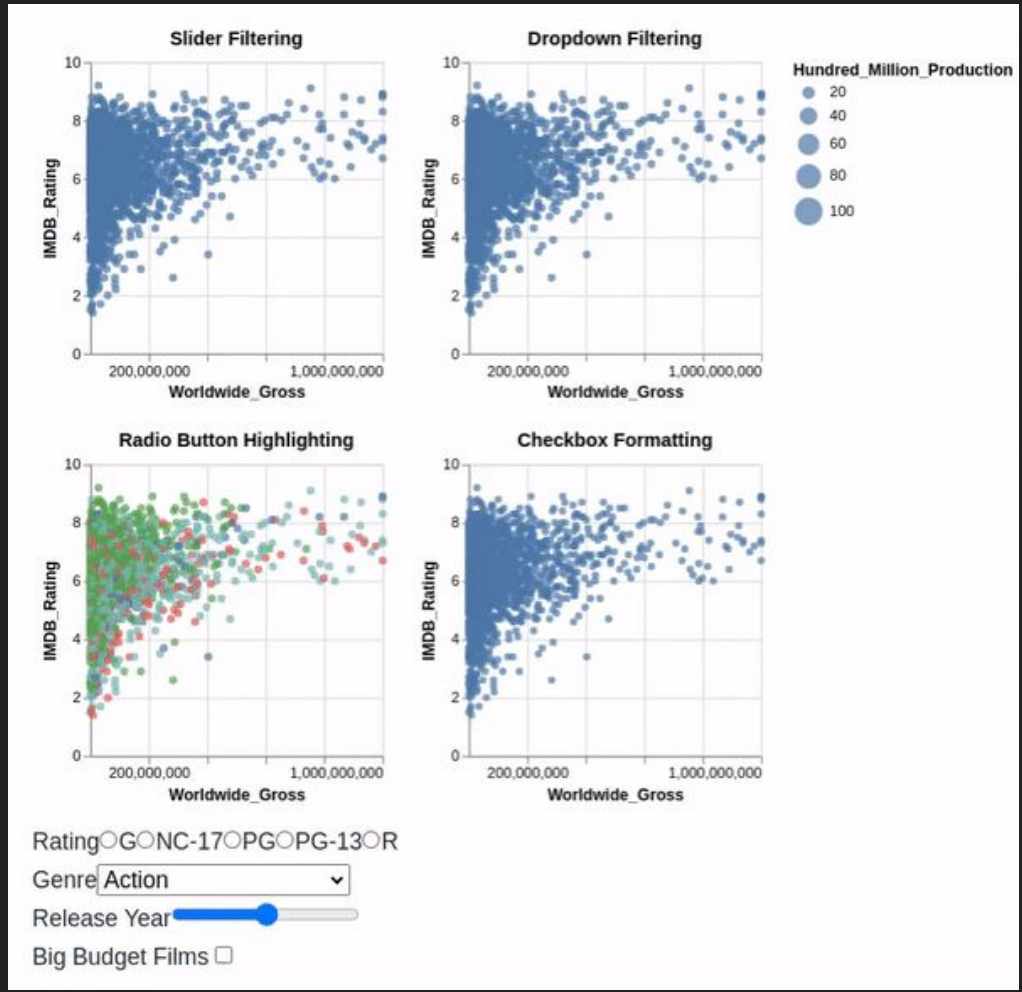
After class:

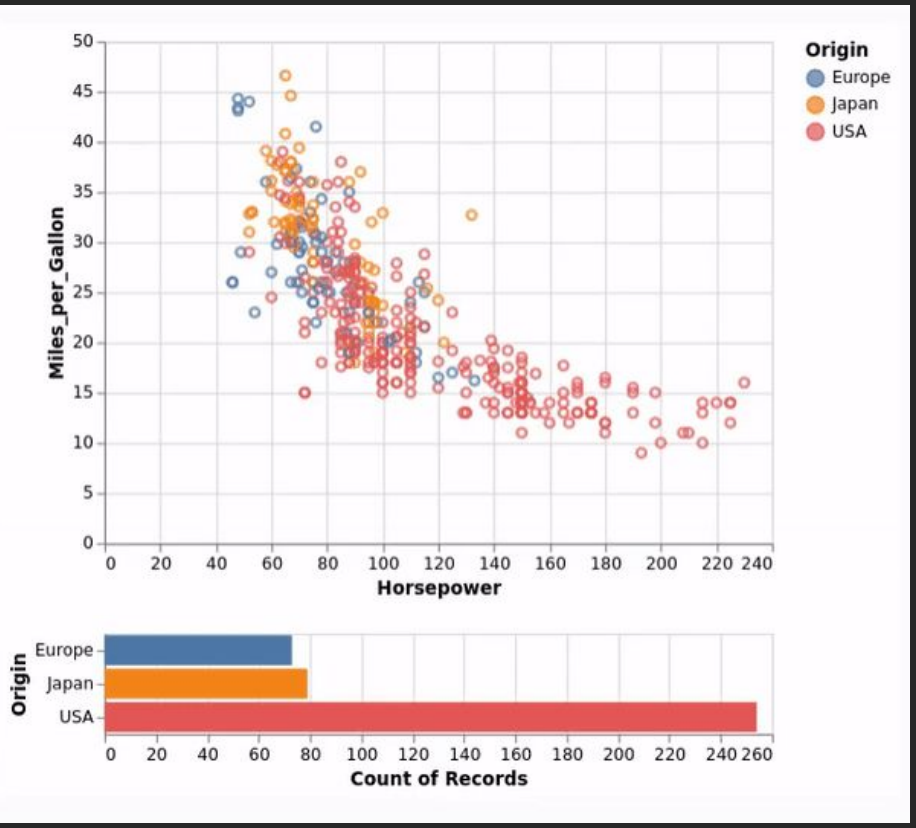
- There are several optional readings linked throughout these notes where you can learn more about Dash and Shiny.

Dashboard technologies

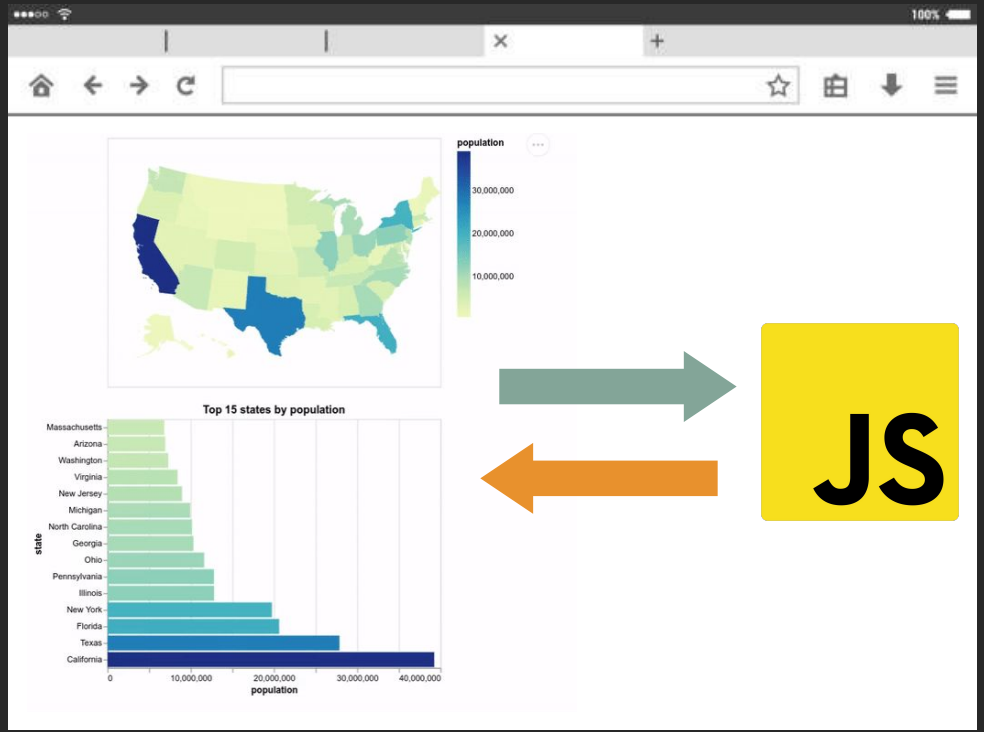

population
30,000,000
20,000,000
10,000,000


Top 15 states by population
Massachusetts
Arizona
Washington
Virginia
New Jersey
Michigan
North Carolina
Georgia
Ohio
Pennsylvania
Illinois
New York
Florida
Texas
California
state
0
10,000,000
20,000,000
30,000,000
40,000,000
population

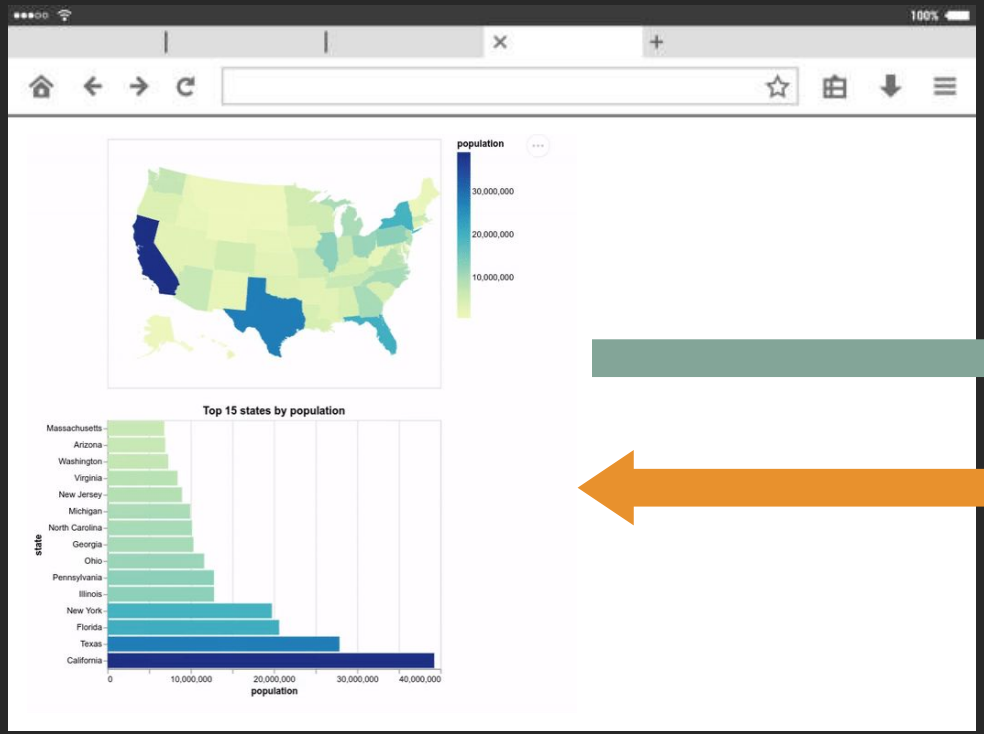

Slider Filtering
Dropdown Filtering
Hundred_Million_Production
20
40
60
80
100
IMDB_Rating
Worldwide_Gross
200,000,000
1,000,000,000
Radio Button Highlighting
Checkbox Formatting
Rating G NC-17 PG PG-13 R
Genre Action
Release Year
Big Budget Films


Origin
Europe
Japan
USA
Miles_per_Gallon
Horsepower
Origin
Europe
Japan
USA
Count of Records

Client side interactivity



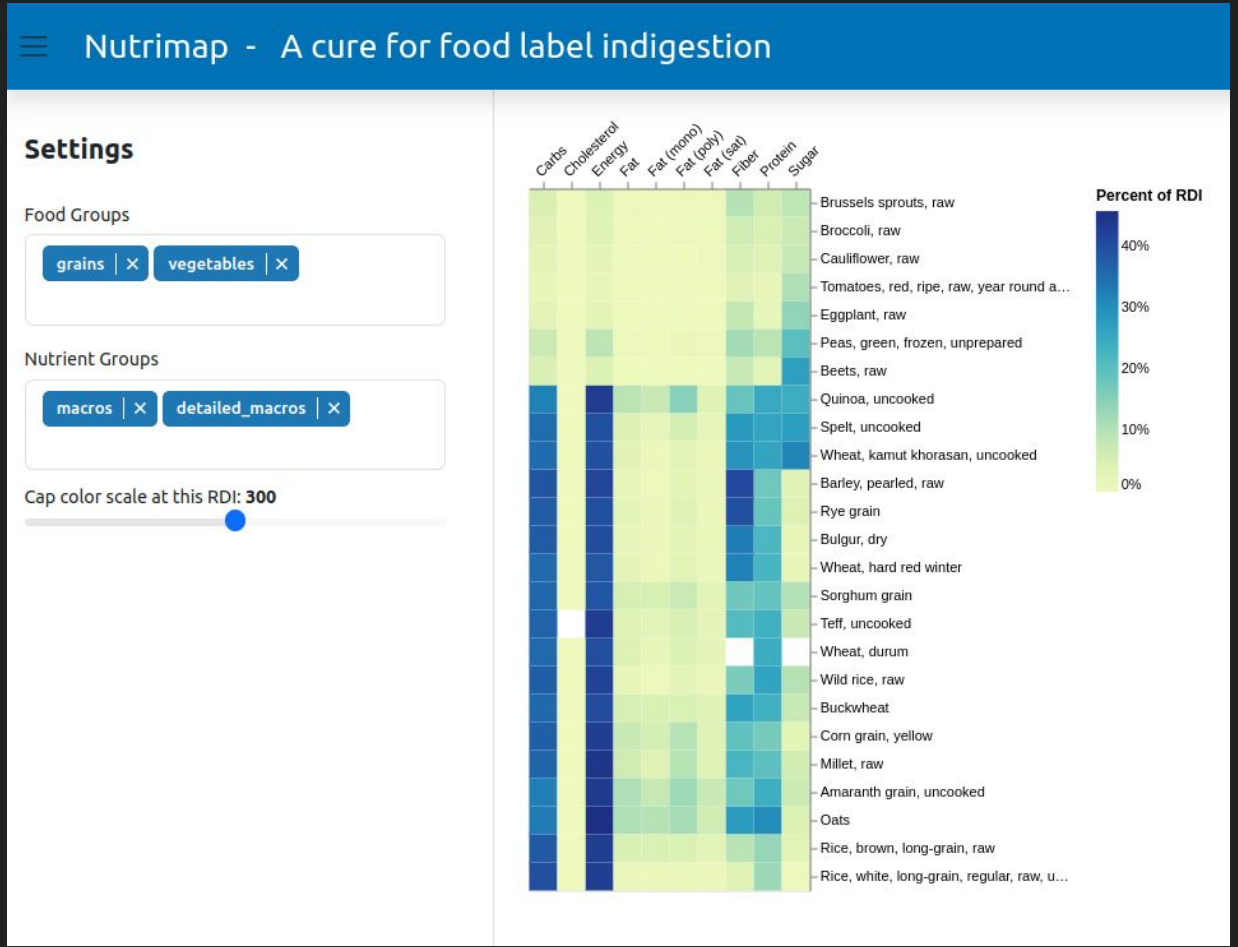
Server side interactivity



Client side vs Server side interactivity

- Client side code runs in your browser locally
  - “Front end”
  - Executes codes locally on the client
    - Only supports code that the browser can execute (JavaScript, WebAssembly/Wasm)
  - + Portable and simple
  - - Not as powerful/versatile, e.g. not many great data science JS libraries
- Server side code runs on a computer with the required languages installed (the server)
  - Usually a computer on the web (the “cloud”)
  - “Back end”
  - Executes code on a server and sends results to a client (e.g. your browser)
  - Supports any programming language that is installed on the server
  - + Powerful and versatile
  - - Setup more involved, often slower, can be expensive for good performance

Server side languages in the browser



New innovations like PyScript/WebR allows traditional server side languages to be executed in browsers (by compiling to Webassembly).

You can e.g. have a fully interactive dashboards running for free on GitHub Pages instead of paying for a server.

Although this is a parenthesis in terms of 532, you can learn more, you can e.g. check out: to docs for Shinylive and Panel to understand how it works (another dashboard library).

Which libraries are we going to use?

Dashboard frameworks in R

Flexible and powerful for making production quality dashboards

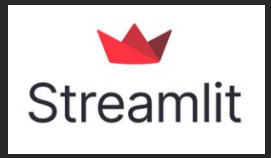
You only work in R, no web stack knowledge needed

Enterprise support available

Dashboard frameworks in Python

- Flexible and powerful for making production quality dashboards
- You only work in Python, no web stack knowledge needed
- Enterprise support available

Still in early stages



Great for building simpler dashboards
Really quick to get started

Turn a notebook into a simple dashboard

Great for quickly sharing ML apps

Allows for both simpler and more complex dashboard. Widgets can be used inside notebooks as well

Traditional web Frameworks targeting more than data dashboards

Other dashboard frameworks

GUI driven

Lower threshold to start using

Less flexible and powerful

Generally expensive

https://hfboyce.github.io/tableau_course/lessons/tableau1.html

Shiny & Dash app basics

Dashboard = Web/Data App(lication)

Easy web apps for data science without the compromises

No web development skills required

Low-Code Python Data Apps

Dashboard/App elements

A dashboard consists of input and output elements

- Input elements (“widgets”) are components like dropdown, sliders, etc.
  - These are provided by the dashboard library

- Output elements are components like tables, charts, etc
  - Tables and text areas are often provided by the dashboard library
  - Charts are provided by external libraries (ggplot, altair, plotly, etc)

Two parts of each app - Frontend & Backend

- Frontend (User interface, UI)
  - Describes the layout of input and output elements in the app
    - E.g. Position widgets in a panel to the left and charts and tables to the right in the dashboard

- Backend (Server)
  - Describes the logic of what happens when you interact with the input elements
    - E.g. Which code is executed when a selection is made in a dropdown? Do any output elements need to be recalculated/replotted with the new input values?
  - This is also called “callbacks” (in Dash) or “reactivity” (in Shiny)

Create an MVP; then iterate