Library Overview

Let’s start with Python’s Dash. It is a Python framework used for building web applications. It’s written in F

Here’s an example dashboard you can create with Dash:

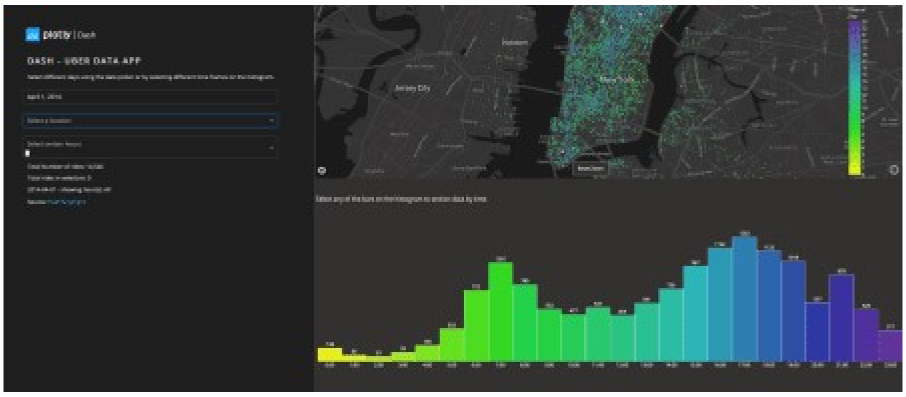


Image 1 – New York Uber Rides Dash Dashboard

> **Want to see more dashboard examples? Check out Plotly’s official app gallery.**

On the other hand, R Shiny is an open-source package for building web applications with R. It provides a

Here’s an example dashboard you can create with Shiny:

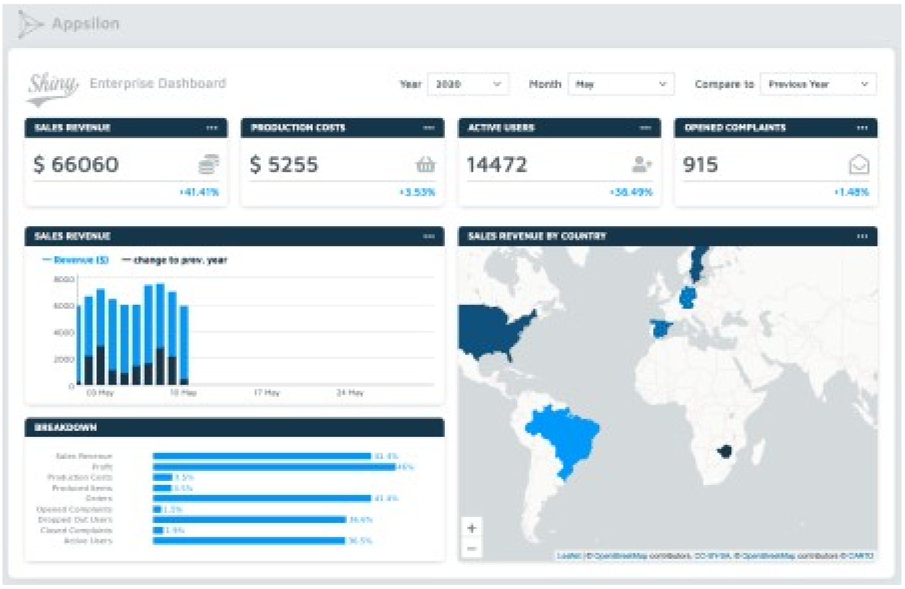


Image 2 – Shiny Enterprise Dashboard

> **What else can you do with Shiny? Here’s our curated collection of demo Shiny dashboards.**

Winner: Tie. You can develop identical solutions with both Dash and Shiny.

Boilerplate Comparison

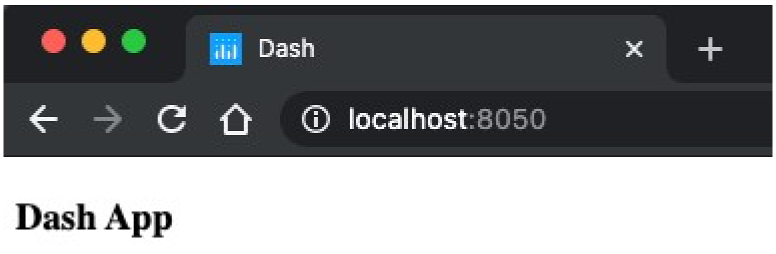
Every web framework comes with a chunk of code needed for the application to run. You can’t avoid writi
H3. Let’s begin with Dash.

```
import dash
import dash_html_components as html

app = dash.Dash(__name__)
app.layout = html.Div(children=[
    html.H3('Dash App')
])


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

Here's the corresponding application:



Dash App

Image 3 – Basic Dash application

So eleven lines in total, and you haven't imported any data visualization library. Three lines are empty – u

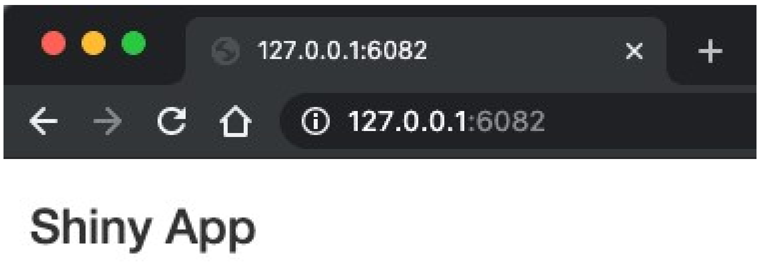
```
library(shiny)

ui <- fluidPage(
  tags$h3("Shiny App")
)

server <- function(input, output) { }

shinyApp(ui, server)
```

Only nine lines here, of which three are empty. Here's the corresponding application:



Shiny App

Image 4 – Basic Shiny application

Winner: R Shiny. Does it really matter much, though? It's only boilerplate code, after all. At this initial sta
ease-of-use throughout the article.

Creating UI Elements

Let’s continue our comparison by taking a look at UI elements. The goal is to create the same form-basec

Let’s start with Python’s Dash. All of the core UI components are available in the `dash_core_componer`

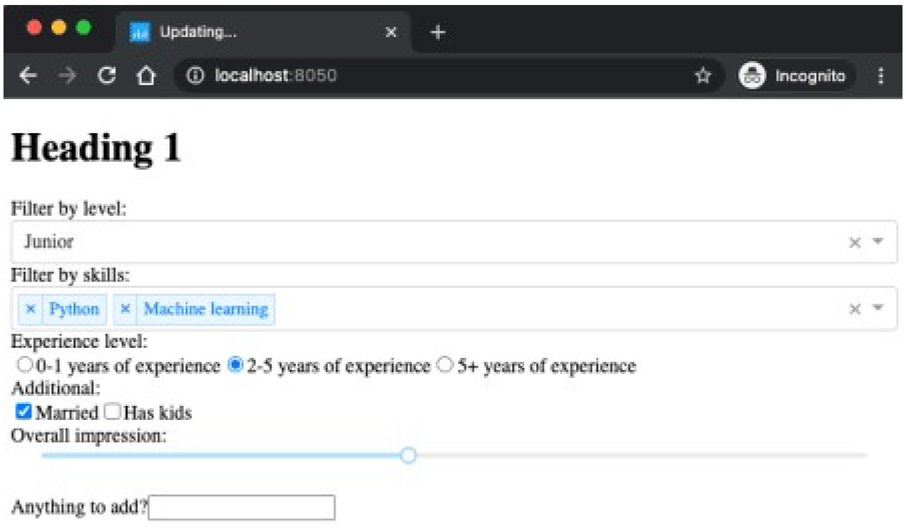
Here’s the code for a simple form-based application:

```
import dash
import dash_core_components as dcc
import dash_html_components as html

app = dash.Dash(__name__)
app.layout = html.Div(children=[
    html.H1('Heading 1'),
    html.Label('Filter by level:'),
    dcc.Dropdown(
        options=[
            {'label': 'Junior', 'value': 'junior'},
            {'label': 'Mid level', 'value': 'mid'},
            {'label': 'Senior', 'value': 'senior'}
        ],
        value='junior'
    ),
    html.Label('Filter by skills:'),
    dcc.Dropdown(
        options=[
            {'label': 'Python', 'value': 'python'},
            {'label': 'R', 'value': 'r'},
            {'label': 'Machine learning', 'value': 'ml'}
        ],
        value=['python', 'ml'],
        multi=True
    ),
    html.Label('Experience level:'),
    dcc.RadioItems(
        options=[
            {'label': '0-1 years of experience', 'value': '01'},
            {'label': '2-5 years of experience', 'value': '25'},
            {'label': '5+ years of experience', 'value': '5plus'}
        ],
        value='25'
    ),
    html.Label('Additional:'),
    dcc.Checklist(
        options=[
            {'label': 'Married', 'value': 'married'},
            {'label': 'Has kids', 'value': 'haskids'}
        ],
        value=['married']
    ),
    html.Label('Overall impression:'),
    dcc.Slider(
        min=1,
        max=10,
        value=5
    ),
    html.Label('Anything to add?'),
    dcc.Input(type='text')
])


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

And here's the corresponding application:



Heading 1

Filter by level:
Junior

Filter by skills:
× Python × Machine learning

Experience level:
○ 0-1 years of experience ● 2-5 years of experience ○ 5+ years of experience

Additional:
☑ Married ☐ Has kids

Overall impression:

Anything to add?

Image 5 – Simple form-based application in Python's Dash

Yeah, not the prettiest. Dash doesn't include too many styles by default, so you'll have to do it independe

Let's replicate the same application with R and Shiny:

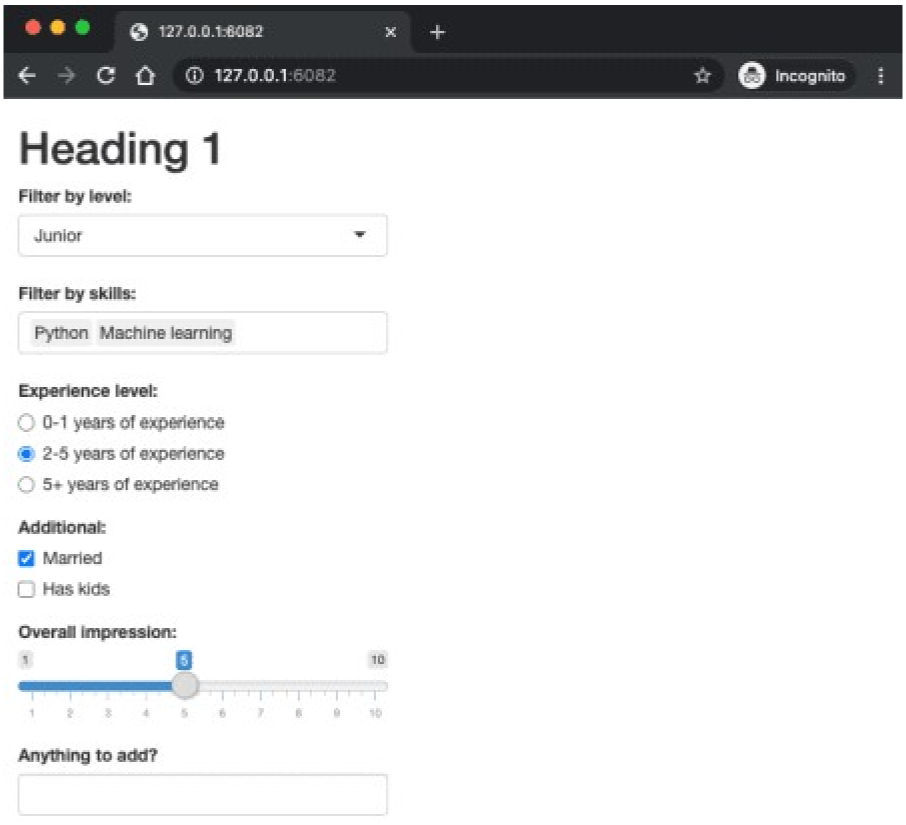
```
library(shiny)

ui <- fluidPage(
  tags$h1("Heading 1"),
  selectInput(
    inputId = "selectLevel",
    label = "Filter by level:",
    choices = c("Junior", "Mid level", "Senior"),
    selected = c("Junior")
  ),
  selectInput(
    inputId = "selectSkills",
    label = "Filter by skills:",
    choices = c("Python", "R", "Machine learning"),
    selected = c("Python", "Machine learning"),
    multiple = TRUE
  ),
  radioButtons(
    inputId = "radioExperience",
    label = "Experience level:",
    choices = c("0-1 years of experience", "2-5 years of experience", "5+ yea
experience"),
    selected = c("2-5 years of experience")
  ),
  checkboxGroupInput(
    inputId = "cbxAdditional",
    label = "Additional:",
    choices = c("Married", "Has kids"),
    selected = c("Married")
  ),
  sliderInput(
    inputId = "slider",
    label = "Overall impression:",
    value = 5,
    min = 1,
    max = 10
  ),
  textInput(
    inputId = "textAdditional",
    label = "Anything to add?"
  )
)

server <- function(input, output) { }

shinyApp(ui, server)
```

Here's the corresponding application:



Heading 1

Filter by level:

Junior

Filter by skills:

Python Machine learning

Experience level:

0-1 years of experience
[✓ 2-5 years of experience]
5+ years of experience

Additional:

[✓ Married]
Has kids

Overall impression:

1 5 10

Anything to add?

Image 6 – Simple form-based application in R Shiny

As you can see, Shiny includes a ton more styling straight out of the box. Shiny applications look better tl

Winner: R Shiny. You can create a better-looking application with less code.

Styling UI with Custom CSS

Nobody likes a generic-looking application. The aesthetics of your app are tied directly with how users fee

Still, adding a touch of syle through CSS is more or less a must for your app. This section compares how

A CSS file for the Dash application goes to the `assets` folder and in the `www` folder for Shiny apps. Creat

Here's the complete CSS for both Dash and Shiny – called `main.css`:

```
@import url('https://fonts.googleapis.com/css2?family=Lato:wght@400;700&display=swap');

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Lato', sans-serif;
}

.wrapper {
    padding: 1rem 2rem;
    background-color: #f2f2f2;
    height: 100vh;
}

.main-title {
    font-size: 3rem;
}

.paragraph-lead {
    font-size: 1.25rem;
    color: #777777;
}

.card {
    background-color: #ffffff;
    margin-top: 1.5rem;
    padding: 1rem 1.25rem;
    border: 0.1rem solid #c4c4c4;
    border-radius: 0.3rem;
    min-height: 12rem;
}

.card-title {
    margin-bottom: 1rem;
}
```

```
.card-text {
    margin-bottom: 3.5rem;
}

.card-button {
    background-color: #0099f9;
    color: #ffffff;
    font-weight: bold;
    border: 0.2rem solid #0099f9;
    border-radius: 0.5rem;
    padding: 1rem 1.5rem;
    cursor: pointer;
    transition: all 0.15s;
}

.card-button:hover {
    background-color: #ffffff;
    color: #0099f9;
}
```

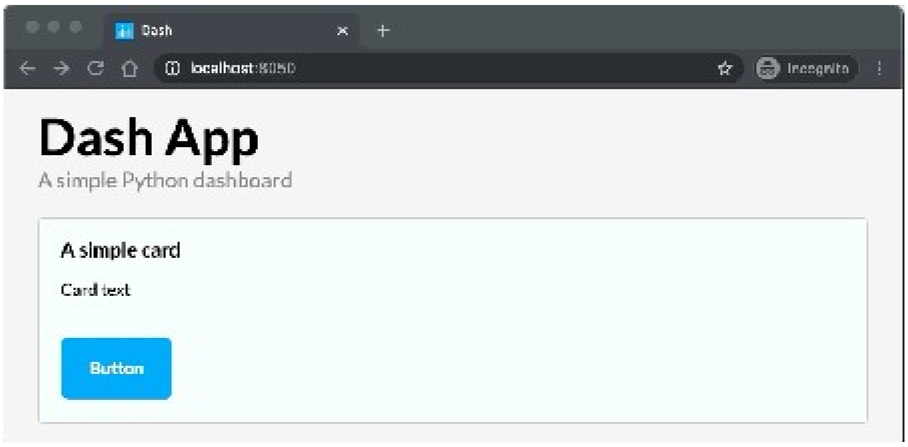
Let’s now use it to create a simple styled application – first with Dash:

```
import dash
import dash_html_components as html

app = dash.Dash(__name__)
app.layout = html.Div(
    className='wrapper',
    children=[
        html.H3('Dash App', className='main-title'),
        html.P('A simple Python dashboard', className='paragraph-lead'),
        html.Div(
            className='card',
            children=[
                html.H3('A simple card', className='card-title'),
                html.P('Card text', className='card-text'),
                html.A('Button', className='card-button')
            ]
        )
    ])


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

Here’s the corresponding application:



Dash App

A simple Python dashboard

A simple card

Card text

Button

Image 7 – Styled Dash application

As you can see, the stylings work just as for any regular web application. That's because Dash didn't add

Here's the code for a styled R Shiny app:

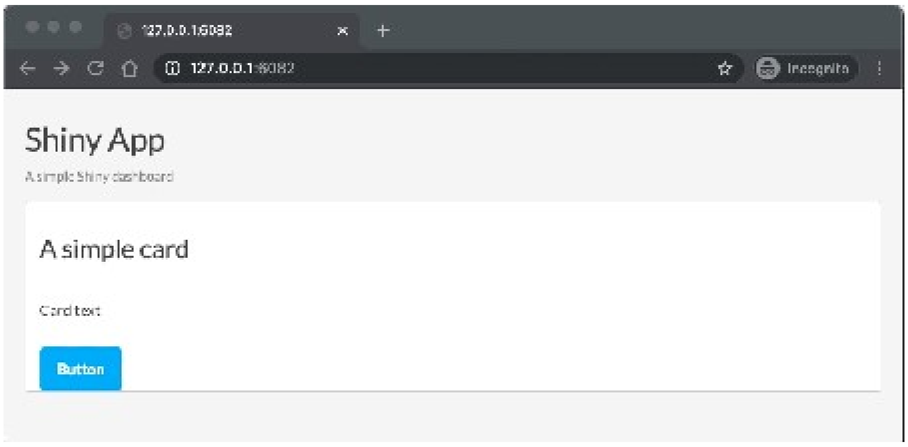
```
library(shiny)

ui <- fluidPage(
  theme = "main.css",
  class = "wrapper",
  tags$h3("Shiny App", class = "main-title"),
  tags$p("A simple Shiny dashboard", class = "paragraph-lead"),
  tags$div(
    class = "card",
    tags$h3("A simple card", class = "card-text"),
    tags$p("Card text", class = "card-text"),
    tags$a("Button", class = "card-button")
  )
)

server <- function(input, output) { }

shinyApp(ui, server)
```

And here's the corresponding dashboard:



Shiny App

A simple Shiny dashboard

A simple card

Card text

Button

Image 8 – Styled Shiny application

Well, that obviously doesn't look right. The more elaborate default Shiny styling is conflicting with our cus
basic level.

Winner: Dash. This doesn't mean you can't make Shiny apps look fantastic, though.

Styling UI with Bootstrap

Let's discuss CSS frameworks. Bootstrap's been one of the most popular frameworks for years, so natur
"standard" appearance. Anyone who has been in web design/development for more than a short period k

Including Bootstrap in Dash is easy. You just have to install the `dash_bootstrap_components` library

```
import dash
import dash_html_components as html
import dash_bootstrap_components as dbc

app = dash.Dash(
    external_stylesheets=[dbc.themes.BOOTSTRAP]
)
navbar = dbc.NavbarSimple(
    children=[
        dbc.NavItem(dbc.NavLink('About', href='#')),
        dbc.NavItem(dbc.NavLink('Products', href='#')),
        dbc.DropdownMenu(
            children=[
                dbc.DropdownMenuItem('More pages', header=True),
                dbc.DropdownMenuItem('Page 2', href='#'),
                dbc.DropdownMenuItem('Page 3', href='#'),
            ],
            nav=True,
            in_navbar=True,
            label='More',
        ),
    ],
    brand='Bootstrap NavBar',
    brand_href='#',
    color='#0099f9',
    dark=True,
)

jumbotron = dbc.Jumbotron([
    html.H1('Welcome', className='display-3'),
    html.P('This is a sample jumbotron', className='lead'),
    html.Hr(className='my-2'),
    html.P('Additional dummy text'),
    html.P(dbc.Button('Learn more', color='primary'), className='lead'),
])

app.layout = html.Div([
    navbar,
    jumbotron
])
```

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

Here's the corresponding application:

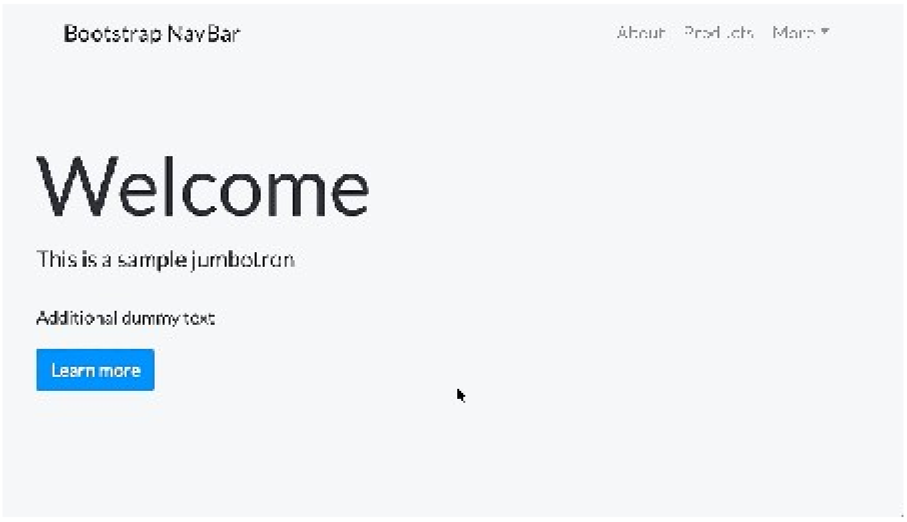


Image 9 – Dash and Bootstrap

R Shiny is different. It comes with Bootstrap out of the box, but not with the most recent version. Bootstra

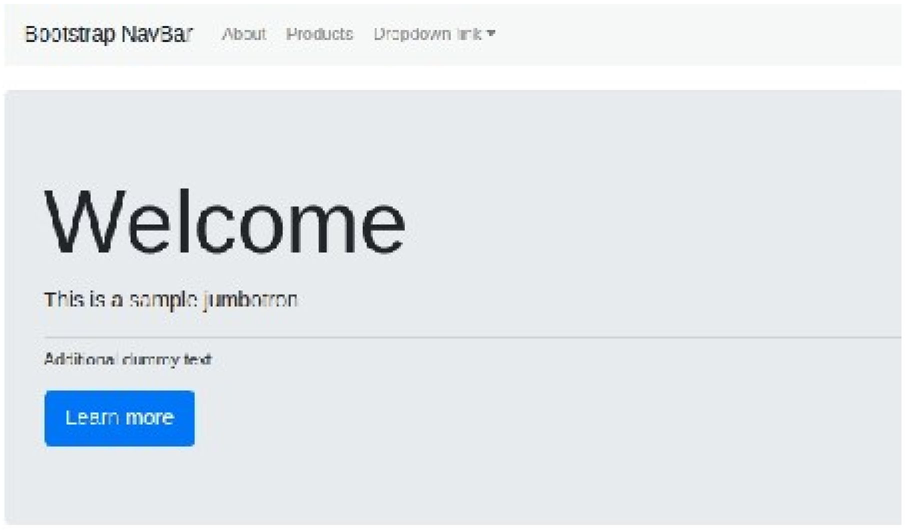


Image 10 – Shiny and Bootstrap

As you can see, it is a code-heavy solution. It requires you to specify every CSS class and other propertie

Winner: Dash. Using Bootstrap is much cleaner in Dash than in Shiny.

Reactivity

Dashboards shouldn't look and behave like plain reports. They have to be interactive. Users won't like yo
components, delay updates until the button is pressed, etc. We'll stick to the basics and perform the upda

For our example, you'll have a single text input and two dropdown menus. Their values determine how th

Let's start with Python and Dash. Here's the code:

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
import statsmodels.api as sm
import pandas as pd

data = pd.DataFrame(sm.datasets.get_rdataset('mtcars', 'datasets', cache=True
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.layout = html.Div([
    html.Label('Plot title:'),
    dcc.Input(id='input-title', value='MTCars visualization', type='text'),
    html.Br(),
    html.Label('X-axis:'),
    dcc.Dropdown(
        id='dropdown-x',
        options=[{'label': col, 'value': col} for col in data.columns],
        value='mpg'
    ),
    html.Br(),
    html.Label('Y-axis:'),
    dcc.Dropdown(
        id='dropdown-y',
        options=[{'label': col, 'value': col} for col in data.columns],
        value='mpg'
    ),
    html.Br(),
    html.Div(
        dcc.Graph(id='chart')
    )
], style={'padding': '30px'})

@app.callback(
    Output('chart', 'figure'),
    Input('input-title', 'value'),
    Input('dropdown-x', 'value'),
    Input('dropdown-y', 'value'))
def update_output(title, x_axis, y_axis):
    fig = px.scatter(data, x=x_axis, y=y_axis, title=title)
    return fig


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

And here's the corresponding dashboard:

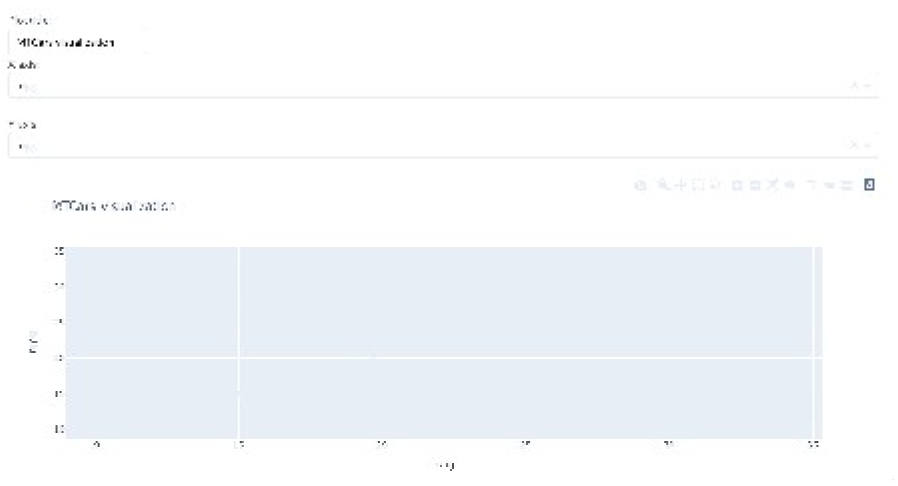


Image 11 – Reactivity demonstration with Python and Dash

It’s a simple dashboard but requires a fair amount of code. Nevertheless, the code is simple to read and u

R Shiny is a bit simpler. It requires significantly less code to produce the same output. Still, the syntax mi

Here’s the code for producing the identical dashboard:

```
library(shiny)
library(ggplot2)
library(plotly)

ui <- fluidPage(
  tags$div(
    class = "wrapper",
    tags$style(type = "text/css", ".wrapper {padding: 30px}"),
    textInput(inputId = "inputTitle", label = "Plot title:", value = "MTCars 
    varSelectInput(inputId = "dropdownX", label = "X-axis", data = mtcars),
    varSelectInput(inputId = "dropdownY", label = "Y-axis", data = mtcars),
    plotlyOutput(outputId = "chart")
  )
)

server <- function(input, output) {
  output$chart <- renderPlotly({
    col_x <- sym(input$dropdownX)
    col_y <- sym(input$dropdownY)

    p <- ggplot(mtcars, aes(x = !!col_x, y = !!col_y)) +
      geom_point() +
      labs(title = input$inputTitle)
    ggplotly(p)
  })
}

shinyApp(ui, server)
```

Here’s the corresponding dashboard:

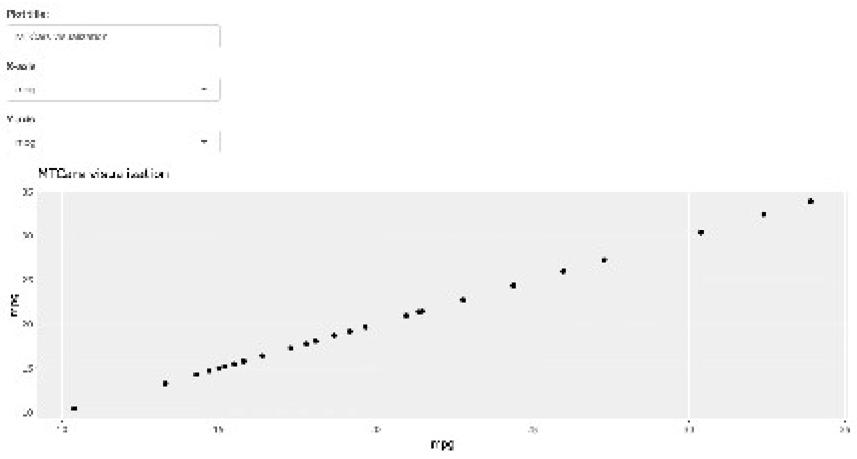


Image 12 – Reactivity demonstration with R Shiny

Winner: R Shiny. Shiny requires less code than Dash for better-looking output.

Conclusion

The final results are in:

- R Shiny – 3 points
- Python Dash – 2 points
- Tie – 1 point

It looks like R shiny is ahead by a single point. Does this mean that R Shiny better for everyone and ever