Introducing semantic.dashboard

Dashboards allow you to structure reports intuitively and break them down into easy-to-read chunks. As a

The `shinydashboard` R package has been out for ages, and it is a good option with a decent amount o

The `semantic.dashboard` package is an open-source alternative to `shinydashboard` created by Ap

For example, let's take a look at two identical applications – the first built with `shinydashboard`, and the

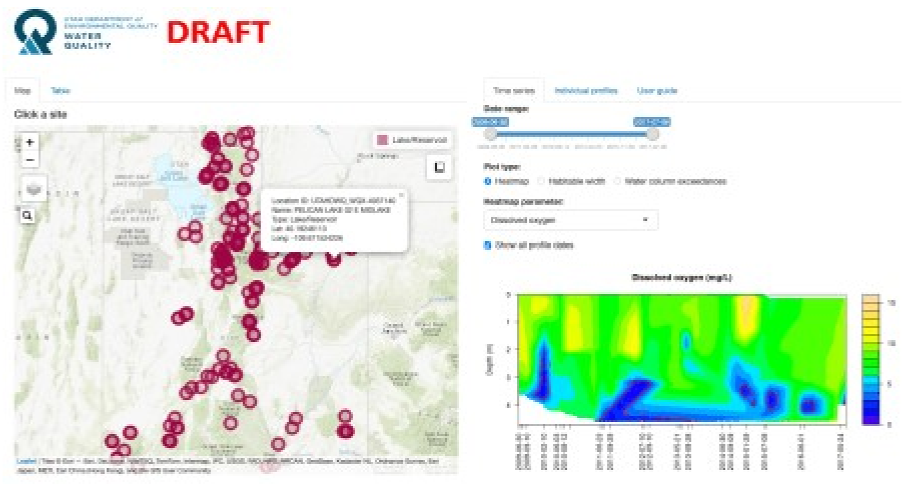


Image 1 – Dashboard built with shinydashboard

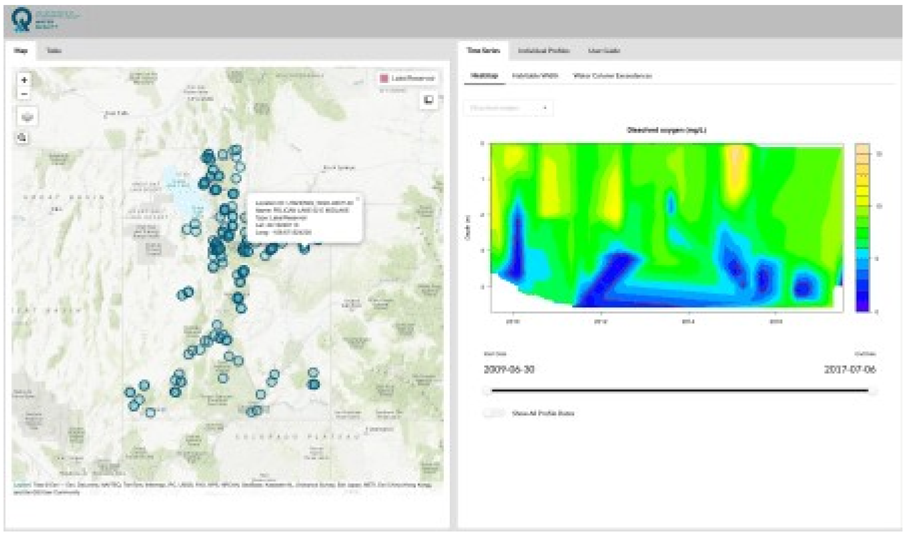


Image 2 – Dashboard built with semantic.dashboard

Both look good – that's guaranteed, but the one built with `semantic.dashboard` doesn't look as generi

You can download the source code of this article here.

Want to learn more about `semantic.dashboard`? Visit the official Github page. Feel free to leave

Navigate to a section:

- semantic.dashboard Installation and Your First Dashboard
- Build a Fully Interactive Dashboard
- Conclusion

To learn more about Appsilon's open-source packages, see our new Open-Source Landing Page:

Appsilon's shiny.tools landing page for our open source packages.

Installation and Your First Dashboard

The `semantic.dashboard` package is available on CRAN (*Comprehensive R Archive Network*). To ins

```
install.packages("semantic.dashboard")
```

You can now proceed by creating an empty dashboard:

```
library(shiny)
library(semantic.dashboard)

ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(sidebarMenu()),
  dashboardBody()
)

server <- function(input, output) { }

shinyApp(ui, server)
```

Here's the corresponding output:

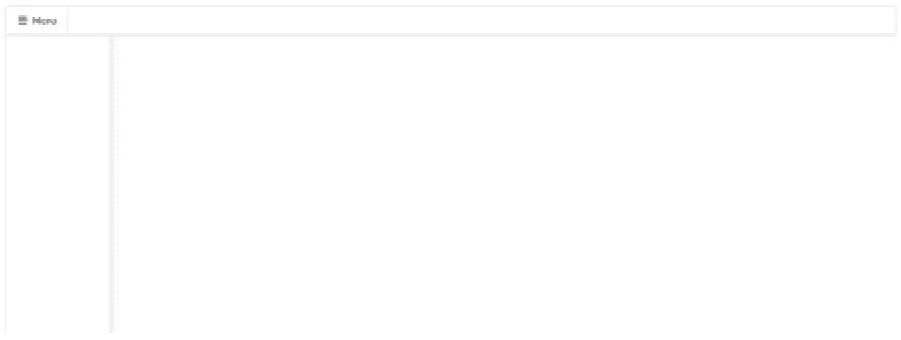


Image 3 – Empty shiny.semantic dashboard

The `semantic.dashboard`'s app UI is made of a `dashboardPage`, which is further split into three eler

1. **Header** – `dashboardHeader`

2. **Sidebar** – `dashboardSidebar`
3. **Body** – `dashboardBody`

This structure is identical as with `shinydashboard` – making things easier to learn. Let's see how to twe

There are a lot of things you can do with `dashboardHeader`. For example, you can change the color by

Here's how to change the color from white to something less boring:

```
dashboardHeader(color = "blue", inverted = TRUE)
```

The `inverted` parameter sets the color to the header background instead of the header text.

Here's the corresponding output:

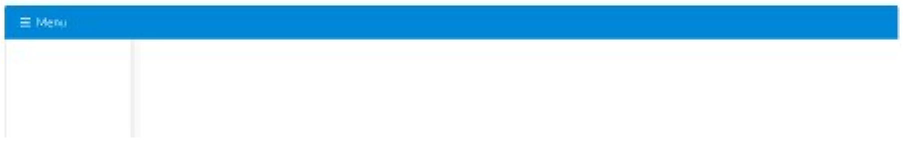


Image 4 – Styling the header of the semantic dashboard

Next, let's see how to add elements to the `dashboardSidebar`. You can specify the sidebar size by twe

Here's how to make the sidebar wider:

```
dashboardSidebar(
  size = "wide",
  sidebarMenu(
    menuItem(tabName = "panel1", text = "Panel 1"),
    menuItem(tabName = "panel2", text = "Panel 2")
  )
)
```

Here are the results:

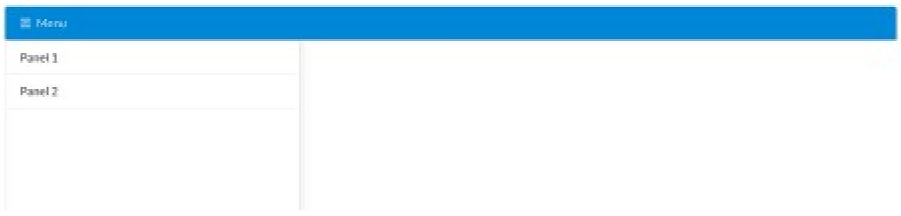


Image 5 – Styling the sidebar of the semantic dashboard

That adds the elements to the sidebar, but how can you display different content when the tab is clicked?

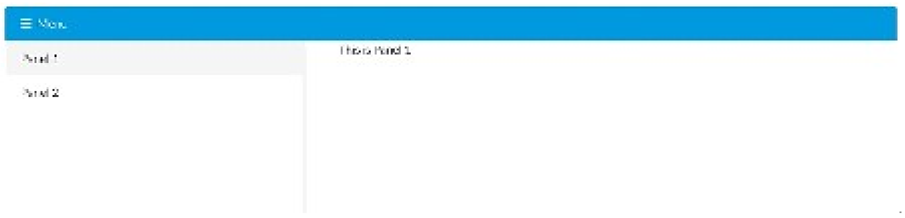
Let's add `tabItems` and two tabs, corresponding to two options in the sidebar. The first option is selecte

```
dashboardBody(
  tabItems(
    selected = 1,
    tabItem(
      tabName = "panel1",
      textOutput(outputId = "text1")
    ),
    tabItem(
      tabName = "panel2",
      textOutput(outputId = "text2")
    )
  )
)
```

To make this work, you'll need to make some tweaks to the `server` function. You'll have to render text or

```
server <- function(input, output) {
  output$text1 <- renderText("This is Panel 1")
  output$text2 <- renderText("This is Panel 2")
}
```

Your dashboard should look like this now:



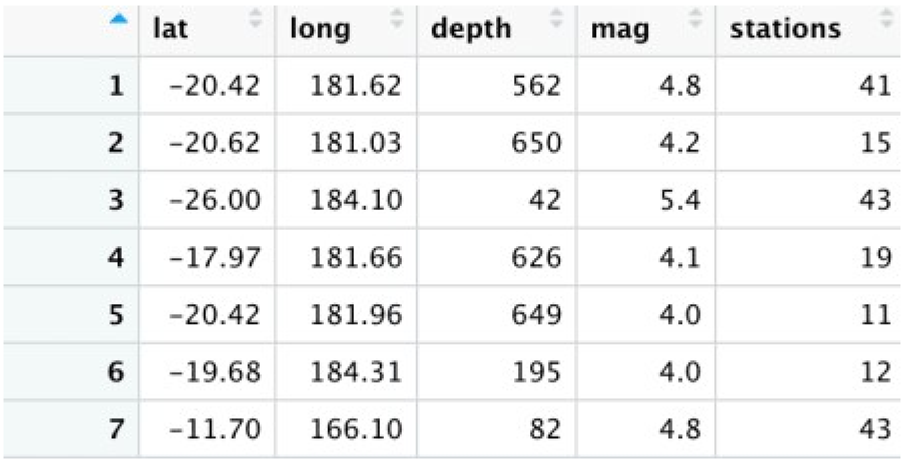
This is Panel 1

Image 6 – Initial dashboard with two panels

Now you know the basics of `semantic.dashboard`. Let's see how to take it a step further and display a

Build a Fully Interactive Dashboard

R comes with a lot of built-in datasets, `quakes` being one of them. It shows geolocations of 1000 seismic



	lat	long	depth	mag	stations
1	-20.42	181.62	562	4.8	41
2	-20.62	181.03	650	4.2	15
3	-26.00	184.10	42	5.4	43
4	-17.97	181.66	626	4.1	19
5	-20.42	181.96	649	4.0	11
6	-19.68	184.31	195	4.0	12
7	-11.70	166.10	82	4.8	43

Image 7 – First couple of rows of the Quakes dataset

You'll now see how to develop a semantic dashboard with the following tabs:

- **Interactive map** – display geographical area near Fiji with markers representing the magnitude of
- **Table** – shows the source dataset formatted as a table

You'll create the interactive map with the `leaflet` package, so make sure to have it installed:

```
install.packages("leaflet")
```

The UI follows the pattern discussed in the previous section – there's a header, sidebar, and a body. The

- `leafletOutput()` – used to display the interactive map
- `dataTableOutput()` – used to display the data table

To make the map as large as possible, you can set some inline CSS styles. In the code below, the height

Here's the code for the UI:

```
library(shiny)
library(shiny.semantic)
library(shinydashboard)
library(leaflet)

ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(
    size = "wide",
    sidebarMenu(
      menuItem(tabName = "map", text = "Map", icon = icon("map")),
      menuItem(tabName = "table", text = "Table", icon = icon("table"))
    )
  ),
  dashboardBody(
    tabItems(
      selected = 1,
      tabItem(
        tabName = "map",
        tags$style(type = "text/css", "#map {height: calc(100vh - 80px) !impo
        leafletOutput("map")
      ),
      tabItem(
        tabName = "table",
        fluidRow(
          h1("Quakes Table"),
          semantic_DTOutput("quakesTable")
        )
      )
    )
  )
)
```

In order to make this dashboard work, you'll have to modify the `server` function. Inside it lies the code fc

The magnitude of the seismic activity determines the size of a marker. Every marker is clickable – showir

Here's the code for the server:

```
server <- function(input, output) {
  output$map <- renderLeaflet({ leaflet() %>%
      setView(lng = 179.3355929, lat = -20.4428959, zoom = 6.5) %>%
      addProviderTiles("Esri.WorldStreetMap") %>%
      addCircles(
        data = quakes,
        radius = sqrt(10^quakes$mag) * 30,
        color = "#000000",
        fillColor = "#ffffff",
        fillOpacity = 0.5,
        popup = paste0(
          "Magnitude: ", quakes$mag, "
",
          "Depth (km): ", quakes$depth, "
",
          "Num. stations reporting: ", quakes$stations
        )
      )
  })

  output$quakesTable <- DT::renderDataTable(
    semantic_DT(quakes)
  )
}
```

And here’s the final dashboard: