

Shiny App




<https://wdt1w3-alondra0leilany0delgado.shinyapps.io/stationsHourly/>


Este código está destinado a implementar una interfaz gráfica Shiny en R con el fin de visualizar datos relacionados con la contaminación atmosférica en México provenientes de 73 estaciones diferentes que recopilamos datos desde 2011 hasta 2021 sobre la concentración de 23 contaminantes atmosféricos, temperatura y humedad relativa. Por la naturaleza del dataset se seleccionaron aquellos contaminantes que poseen mejor densidad en sus registros y proporcionaron una mejor oportunidad de análisis. La contaminación del aire es una de las principales causas de muertes prematuras en todo el mundo y a menudo es insuficientemente monitoreada en países en desarrollo. Esperamos que esta interfaz gráfica pueda llamar la atención sobre la necesidad de abordar más legislación en relación con este problema.

Código en R

Estructura del código

 data

 rsconnect

 server


 ui


Interior de la carpeta del
proyecto


Dentro de la carpeta cdmxStation se encuentran los archivos:


server.R 1. Contiene la lógica del servidor en la aplicación. Contiene las funciones y operaciones que se ejecutan en el lado del servidor cuando los usuarios interactúan con la aplicación.


ui.R Contiene el código que define la interfaz de usuario (UI) de la aplicación y establece los lugares donde se desplegarán elementos generados en server.


 **rsconnect** Contiene elementos generados por Shiny para el despliegue online de la aplicación.


 dict_info

 stations_daily

 stations_hourly

 stations_hourly_clean

 stations_rsinaica

 stations_rsinaica_clean

Interior carpeta data

 **data** Contiene los archivos en la aplicación.

dict_info.R Archivo de R que contiene información en

diccionarios que se despliega en la app

stations_daily.csv

stations_hourly.csv

stations_hourly_clean.csv

stations_rsinaica.csv

stations_rsinaica_clean.csv

Librerías usadas

 **ui.R**

```
library(shiny)
library(DT)
library(shinydashboard)
```

library(shinydashboard)

Extensión del paquete Shiny que permite crear dashboards interactivos.

library(DT)

Proporciona funciones para crear tablas interactivas.

 **server.R**

```
library(shiny)
library(DT)
library(ggplot2)
library(leaflet)
```

library(ggplot2)

Permite crear una gran variedad de gráficos

library(leaflet)

Facilita la creación de mapas interactivos en R, en este caso por coordenadas.

Maquetación de UI

```
ui <- dashboardPage(
  dashboardHeader(title = "Contaminación atmosférica en México"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Tabla", tabName = "table", icon = icon("table"))
    )
  ),
  dashboardBody(
```

```

    tabItems(
      tabItem(tabName = "table",
        h2("Tabla"),
        fluidRow(
          column(2, checkboxGroupInput("columns", "Selecciona columnas",
            choices = variables_interes, selected = variables_interes)),
          column(10, DT::dataTableOutput("tableData"))
        )
      )
    )
  )
)

```

Se carga el diseño dentro de la variable `ui` que luego se encargará de desplegar.

`dashboardPage()` es parte de `shinydashboard` y ayuda a definir lo que está dentro de ese contenedor padre. Se define donde está el header y sidebar con `dashboardHeader()` y `dashboardSidebar()` respectivamente. Este último encierra en un `sidebarMenu()` que se puede ocultar a la izquierda en toggle list, dentro se deben definir en `menuItem()` las pestañas que se desplegarán con los argumentos de título, nombre de la pestaña para hacer la conexión y finalmente un icon.

En `dashboardBody()` se definen el cuerpo de la aplicación, es decir, el contenido principal. Al ser un dashboard se definen las pestañas que se renderizarán en `tabItems()`, cada pestaña debe ir dentro de `tabItem()` donde se debe definir el nombre de identificador de la pestaña para referenciarla arriba y opcionalmente un título que se muestra en pantalla. Dentro de cada item/pestaña, se puede definir una estructura interna, en este caso con `fluidRow()` estableciendo una interfaz que se adapta a la pantalla siguiendo la estructura establecida en cada `column()` con medidas que suman 12.

Definición del servidor

```

server <- function(input, output) {

  #Resto del código...

}

```

La función `server` define la lógica del lado del servidor de la aplicación, gestionando las interacciones y actualizaciones en respuesta a los eventos del usuario por medio de una función que tiene como placeholders `input` (lo que recibe de información) y `output` (lo que regresará al usuario), con ayuda de esto se definen las variables y funciones que hará el servidor, junto con la interacción que lleva.

Definición de cada tab (Exposición de tabla)

Tabla

Selecciona las columnas que quieres ver

Show entries

Search:

	datetime	station_id	PM2.5	PM10	NOx	O3
1	2015-04-02	32				
2	2015-05-21	32	0.4549667272727273	0.3474	0.0591936	0.02917625
3	2015-05-22	32	0.0778212358974358	0.3893233333333333	0.0191580588235294	0.0519025151515151
4	2015-05-23	32	0.018036371875	0.3877290625	0.019711	0.0210917066666666
5	2015-05-24	32	0.0249352340425531	0.398041914893617	0.0247567391304347	0.0095680888888888
6	2015-05-25	32	0.0262276723404255	0.3940331914893616	0.0264005869565217	0.0066194981395348
7	2015-05-26	32	0.0227527021276595	0.3930580851063829	0.0251237391304347	0.0131108577777777
8	2015-05-27	32	0.022183829787234	0.3955582978723404	0.0206676956521739	0.0190879511111111
9	2015-05-28	32	0.0166242086956521	0.3916310869565217	0.0173527272727272	0.0165787454545454
10	2015-05-29	32	0.0136389702127659	0.3973559574468085	0.0181878260869565	0.0159072222222222

Showing 1 to 10 of 231,592 entries

Previous 2 3 4 5 ... 23,160 Next

```

dashboardBody(
  tabItems( # Contenido de las pestañas
    tabItem(tabName = "table", #Pestaña 1
      h2("Tabla"),
      fluidRow(
        column(2,
          # Inputs para seleccionar las columnas
          checkboxGroupInput("columns",
            "Selecciona las columnas que quieres ver",
            choices = variables_interes,
            selected = variables_interes)
        ),
        column(10, # Columna de ancho 7
          # Tabla filtrada
          DT::dataTableOutput("tableData")
        )
      )
    )
  )
)

```

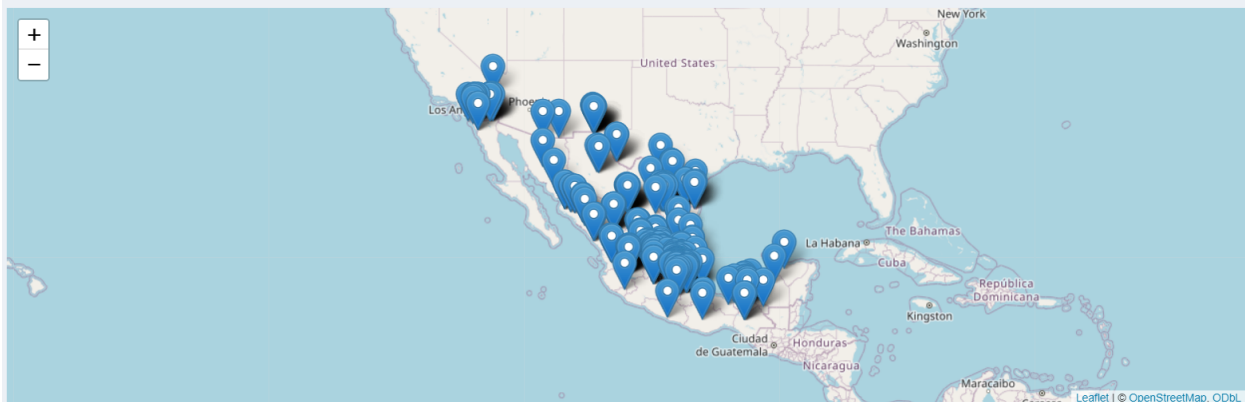
Esto se define dentro de ui, `checkboxGroupInput()` crea un selector con diferentes opciones `choices = variables_interes` para el usuario, marca algunos por default con `selected = variables_interes`. Por último dentro de la otra columna, se renderiza la respuesta del server en `DT::dataTableOutput("tableData")`, los dos puntos son para especificar paqueteria::función

```
# Dentro de variable server
output$tableData <- DT::renderDT({ # Los dos puntos son para especificar paqueteria::función
  subset_table <- datos[, input$columns, drop = F]
  datatable(subset_table, options = list(autoWidth = TRUE))
})
```

El server recibe como datos el input de columns, y `renderDT` toma un bloque de código entre llaves `{ }` que define cómo se debe generar la tabla en función de los eventos o datos específicos del usuario. Se crea una variable llamada `subset_table`, la cual almacena un subconjunto de datos, a partir de lo seleccionando de las columnas especificadas por el usuario en `input$columns` desde el dataset llamado `datos`. `drop = F` asegura que se mantenga la estructura del marco de datos, incluso si solo se selecciona una columna. La función `datatable` convierte el subconjunto de datos `subset_table` en una tabla interactiva. `autoWidth = TRUE` se establece para permitir que la tabla se ajuste automáticamente al ancho de la columna definida en ui. Finalmente este resultado se asigna a `output$tableData` para enviar resultados desde el servidor al ui.

Definición de cada tab (Localización de las estaciones)

Localización de las estaciones



```
tabItem(tabName = "tab2",
  h2("Localización de las estaciones"),
  leafletOutput("map")
)
```

Se define únicamente lo que regresará el server en `leafletOutput("map")` pues no se definió interacción del usuario.

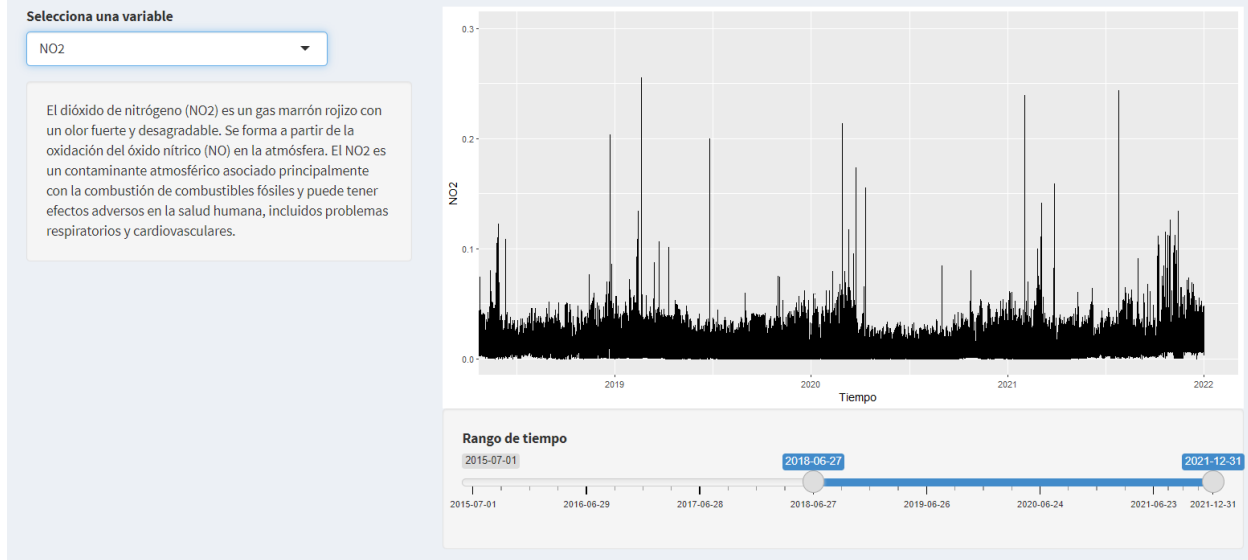
```
output$map <- renderLeaflet({
  mymap <- leaflet() %>% addTiles()

  mymap <- mymap %>%
    addMarkers(
      data = datos_station,
      lng = ~lon,
      lat = ~lat,
      popup = ~paste("Station ID: ", station_id, "<br>Station Name: ", station_name)
    )
  mymap
})
```

Se hace uso de un mapa interactivo mediante la librería Leaflet. La función `renderLeaflet()` asigna el resultado al objeto `output$map` que será mostrado en la interfaz de usuario. El mapa inicializado con `leaflet()` se establece con capas de mapas básicas mediante `addTiles()`. Luego, se añaden marcadores al mapa a partir de los datos contenidos en `datos_station`, especificando las coordenadas de longitud y latitud. Además, se define el contenido emergente de cada marcador, incluyendo información sobre la estación, como su identificador y nombre.

Definición de cada tab (Progreso cronológico de presencia de gases en la atmósfera)

Progreso cronológico de presencia de gases en la atmósfera



```
tabItem(tabName = "tab3",
  h2("Progreso cronológico de presencia de gases en la atmósfera"),
  fluidPage(
    fluidRow(
      column(4, #Parte izquierda
        #INPUT DE VARIABLES
        selectizeInput("gas_select", "Selecciona una variable", choices = names
(dict_gas)),
        wellPanel(
          textOutput("gas_info_render"))
      ),
      column(8,
        verticalLayout(
          plotOutput("timePlot"),
          wellPanel(
            sliderInput("timeSlide",
              "Rango de tiempo",
              min = as.Date("2015-07-01", "%Y-%m-%d"),
              max = as.Date("2021-12-31", "%Y-%m-%d"),
              value = c(as.Date("2015-06-01", "%Y-%m-%d"),
                as.Date("2021-12-31", "%Y-%m-%d")),
              timeFormat = "%Y-%m-%d",
              step = 182)
          )
        )
      )
    )
  ),
),
```

La interfaz de usuario `fluidPage` se divide en dos secciones principales. En la parte izquierda, está `wellPanel` que contiene `selectizeInput` que permite al usuario elegir una variable específica relacionada con los gases atmosféricos seleccionados por tener mejor estructura en sus datos. Estas opciones son dinámicamente generadas a partir de `dict_gas` que es un diccionario que contiene información de cada gas, parte de él hay información que se despliega como texto en `textOutput` según la variable seleccionada. En la parte derecha, ocupando la mayor parte del espacio, se presenta un diseño vertical definido en `verticalLayout`. En esta sección, se incluye un gráfico `plotOutput` que mostrará el progreso cronológico de la variable seleccionada a lo largo del tiempo. Además, se encuentra `wellPanel` que contiene un slider `sliderInput`, permitiendo al usuario ajustar el rango de tiempo que se mostrará en el gráfico. El rango de tiempo inicial se establece entre "2015-06-01" y "2021-12-31", y el deslizador se ajusta en intervalos de 182 días, con el fin de llevar un control de 6 meses.

```
output$timePlot <- renderPlot({
  ggplot(datos_fecha, aes(x = datetime, y = datos_fecha[[input$gas_select]])) +
  labs(x = "Tiempo",
       y = input$gas_select) +
  geom_line() +
  coord_cartesian(xlim = c(input$timeSlide[1], input$timeSlide[2])) +
  scale_y_continuous(limits = c(0, dict_gas[[input$gas_select]]$concent))
})
```

Se utiliza la función `ggplot()` en la que se especifica el dataset `datos_fecha` y se asigna la columna `datetime` al eje x y la variable de contaminante seleccionada `datos_fecha[[input$gas_select]]` al eje y. La función `labs` se utiliza para etiquetar los ejes, utilizando la variable seleccionada tanto para el eje y como para el título del eje y.

Además, se agrega una línea al gráfico mediante `geom_line()`, representando la evolución de la concentración del gas seleccionado. La función `coord_cartesian()` se emplea para limitar el eje x al rango especificado por el slider `input$timeSlide`, permitiendo al usuario ajustar el rango de tiempo. Además, se utiliza `scale_y_continuous()` para establecer límites en el eje y, asegurando que la visualización refleje adecuadamente los valores de concentración del gas atmosférico. Estos límites se determinan a partir de los datos almacenados en `dict_gas` dichos valores son resultado de investigar el dato máximo registrado para esa variable y evitar usar datos erróneos del dataset.

Definición de cada tab (Progreso cronológico de presencia de gases en la atmósfera)

```
tabItem(tabName = "tab4", # Pestaña 2
        h2("Frecuencia en la concentración de cada gas"), # Título de ejemplo
        # -----
        fluidRow(
          column(2,
            #INPUT DE VARIABLES
            selectizeInput("gas_select_2", "Selecciona una variable", choices =
names(dict_gas)),
            # Inputs para seleccionar las variables
            sliderInput("binsHist",
                        "Número de Bins:",
                        min = 1,
                        max = 50,
                        value = 30)
          ),
          column(10, # Columna de ancho 8
            plotOutput("histPlot")
          )
        )
      )
    )
```

`selectizeInput` que permite al usuario elegir una variable específica relacionada con los gases atmosféricos. Estas opciones son generadas a partir de `dict_gas` pues sus keys son cada contaminante. Además, se proporciona un slider `sliderInput` que permite ajustar el número de "bins" (intervalos) para el histograma. A la derecha, se encuentra un área destinada a mostrar el histograma de la variable seleccionada. El resultado se asigna a `output$histPlot` un hist generado `plotOutput`.

```
output$histPlot <- renderPlot({
  histogram <- ggplot(datos_fecha, aes(x = datos_fecha[[input$gas_select_2]])) +
    geom_histogram(bins = input$binsHist, fill = "#ff6029", color = "black") +
    labs(title = paste("Histograma del nivel de ", input$gas_select_2),
         x = paste("Niveles de ", input$gas_select_2),
         y = "Frecuencia") +
    theme_minimal() +
    xlim(min(datos_fecha[[input$gas_select_2]]), dict_gas[[input$gas_select_2]]$concent)

  print(histogram)
})
```

Se utiliza la función `ggplot()` en la que se especifica el conjunto de datos `datos_fecha` y se asigna la variable seleccionada `datos_fecha[[input$gas_select_2]]` al eje x del histograma. `geom_histogram()` se utiliza para generar las barras del histograma, y el número de "bins" se controla mediante el valor seleccionado por el usuario en `input$binsHist`

`labs` se utiliza para agregar títulos y etiquetas a los ejes del histograma, personalizados según la variable seleccionada por el usuario. Se establecen límites en el eje x del histograma utilizando `xlim`, asegurando que el gráfico refleje adecuadamente la concentración del gas atmosférico seleccionado. Estos límites se determinan a partir de la concentración básica definida en `dict_gas`. Finalmente, el histograma se imprime mediante `print(histogram)` para ser mostrado en la interfaz de usuario.