

Digital Health

Covid-19 Data

Use Case:

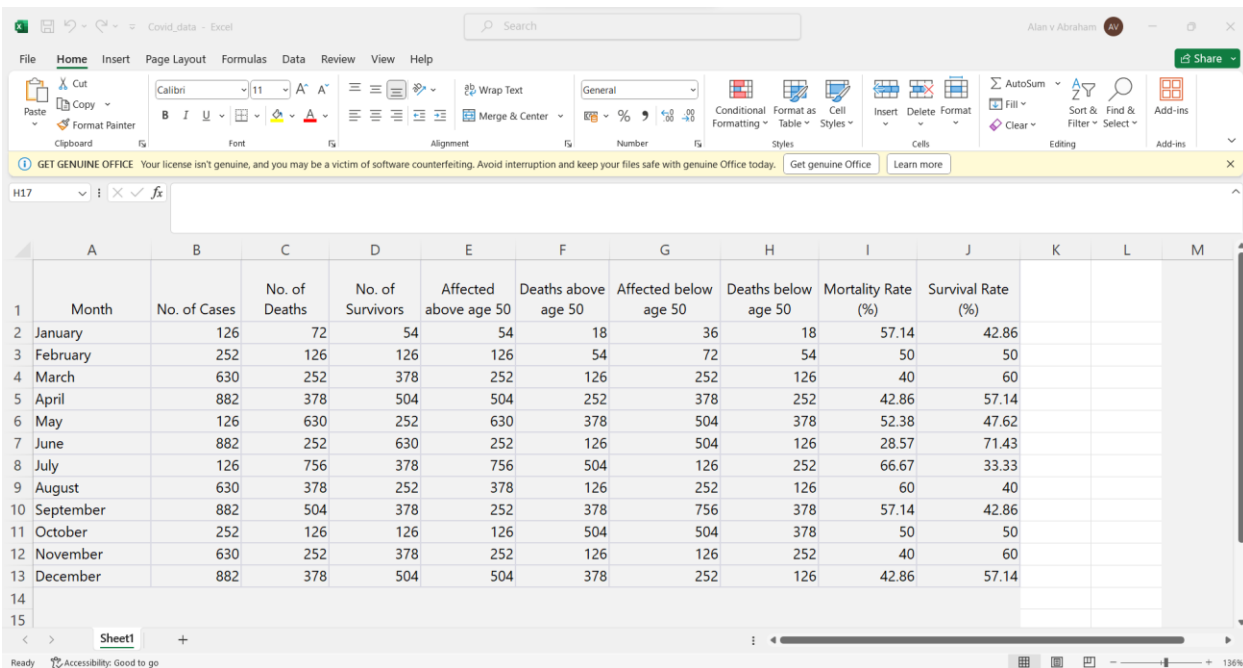
The COVID-19 pandemic has brought about unprecedented challenges worldwide, impacting various aspects of society. Stakeholders involved include healthcare professionals, policymakers, researchers, and the general public. Healthcare professionals need accurate data to make informed decisions about resource allocation, treatment strategies, and public health measures. Policymakers require insights into the spread of the virus to implement effective containment measures and allocate funding appropriately. Researchers rely on data to study the epidemiology of the virus, develop vaccines, and evaluate the effectiveness of interventions. The general public seeks reliable information to understand the severity of the pandemic and adopt preventive measures.

Solution:

To address the challenges posed by the COVID-19 pandemic, a dynamic data visualization tool is developed using R Shiny. The tool allows stakeholders to interactively explore and analyze COVID-19 data, facilitating evidence-based decision-making and public awareness. The pathway to finding a solution involves several key steps:

1. **Data Collection:** Obtaining COVID-19 data from reliable sources such as government health agencies or international organizations.

The dataset includes information on the number of cases, deaths, survivors, demographic characteristics, and other relevant variables.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Month	No. of Cases	No. of Deaths	No. of Survivors	Affected above age 50	Deaths above age 50	Affected below age 50	Deaths below age 50	Mortality Rate (%)	Survival Rate (%)			
2	January	126	72	54	54	18	36	18	57.14	42.86			
3	February	252	126	126	126	54	72	54	50	50			
4	March	630	252	378	252	126	252	126	40	60			
5	April	882	378	504	504	252	378	252	42.86	57.14			
6	May	126	630	252	630	378	504	378	52.38	47.62			
7	June	882	252	630	252	126	504	126	28.57	71.43			
8	July	126	756	378	756	504	126	252	66.67	33.33			
9	August	630	378	252	378	126	252	126	60	40			
10	September	882	504	378	252	378	756	378	57.14	42.86			
11	October	252	126	126	126	504	504	378	50	50			
12	November	630	252	378	252	126	126	252	40	60			
13	December	882	378	504	504	378	252	126	42.86	57.14			

Figure 1. Covid Data

2. **Data Cleaning and Preparation:** Cleaning of the dataset to remove any inconsistencies, missing values, or outliers and performing data transformations and aggregations as needed to prepare the data for visualization.
3. **Development of Shiny App:** The next step is to create a Shiny app with an intuitive user interface that allows stakeholders to interact with the data. Designing the app layout to include input controls for selecting variables, plot types, and other parameters.
4. **Visualization Implementation:** Using **plotly** it generates interactive plots such as bar plots, line plots, and combined plots based on the selected variables. Adding tooltips, hover effects, and other interactive features enhances usability and interpretability.
5. **Testing and Iteration:** Testing the Shiny app extensively to ensure functionality, performance, and usability and Gathering feedback from stakeholders, iterating on the design based on their input. And Address any issues or bugs identified during testing.
6. **Deployment and Maintenance:** Deploying the Shiny app to a web server or cloud platform for public access. Ensures proper maintenance and updates to keep the app running smoothly and address any emerging needs or changes in the data.

Implementation:

1. Data Input:

- **User-Friendly Interface:** Designing a user-friendly interface that allows users to easily upload an Excel file containing the COVID-19 data. Using the fileInput widget in the Shiny app enables file upload functionality.
- **Data Validation:** Implementing data validation checks to ensures that the uploaded file is in the correct format and contains the required fields. Providing error messages or notifications to guide users in case of any issues with the uploaded data.
- **Data Parsing:** Using the read_excel function from the readxl package to parse the uploaded Excel file and read the data into a data frame. Handling any potential errors or exceptions that may occur during the data parsing process.

```
library(rsconnect)
library(shiny)
library(plotly)
library(readxl)
rsconnect::setAccountInfo
```

```
# Read the uploaded Excel file
dataset <- reactive({
  req(input$file)
  inFile <- input$file
  if (is.null(inFile))
    return(NULL)
  df <- read_excel(inFile$datapath)
  df$Month <- factor(df$Month, levels = months_order, ordered = TRUE)
  return(df)
})
```

2. Plot Generation:

- **Modular Approach**: Organizing the code into separate functions for generating different types of plots, such as bar plots, line plots, and combined plots. This modular approach makes the code more maintainable and allows for easier customization of each plot type.
- **Interactive Features**: Incorporating interactive features such as tooltips, hover effects, and zooming to enhance the user experience. Using the `plot_ly` function from the `plotly` package to create interactive plots that respond to user interactions.
- **Customization Options**: Providing customization options for the plots, such as adjusting colors, labels, and axis scales, Allowing users to customize the appearance of the plots based on their preferences and specific visualization needs.

```
# Create bar plot
output$barplot <- renderPlotly({
  req(dataset())
  req(input$variable)

  p <- plot_ly(data = dataset(), x = ~Month, type = 'bar') %>%
    layout(title = paste("Bar Plot of", input$variable),
           xaxis = list(title = "Month"),
           yaxis = list(title = input$variable))

  for (var in input$variable) {
    p <- add_trace(p, y = ~get(var), name = var)
  }

  p
})
```

```

# Create line plot
output$lineplot <- renderPlotly({
  req(dataset())
  req(input$variable)

  p <- plot_ly(data = dataset(), x = ~Month, type = 'scatter', mode = 'lines+markers') %>%
    layout(title = paste("Line Plot of", input$variable),
           xaxis = list(title = "Month"),
           yaxis = list(title = input$variable))

  for (var in input$variable) {
    p <- add_trace(p, y = ~get(var), name = var)
  }

  p
})

```

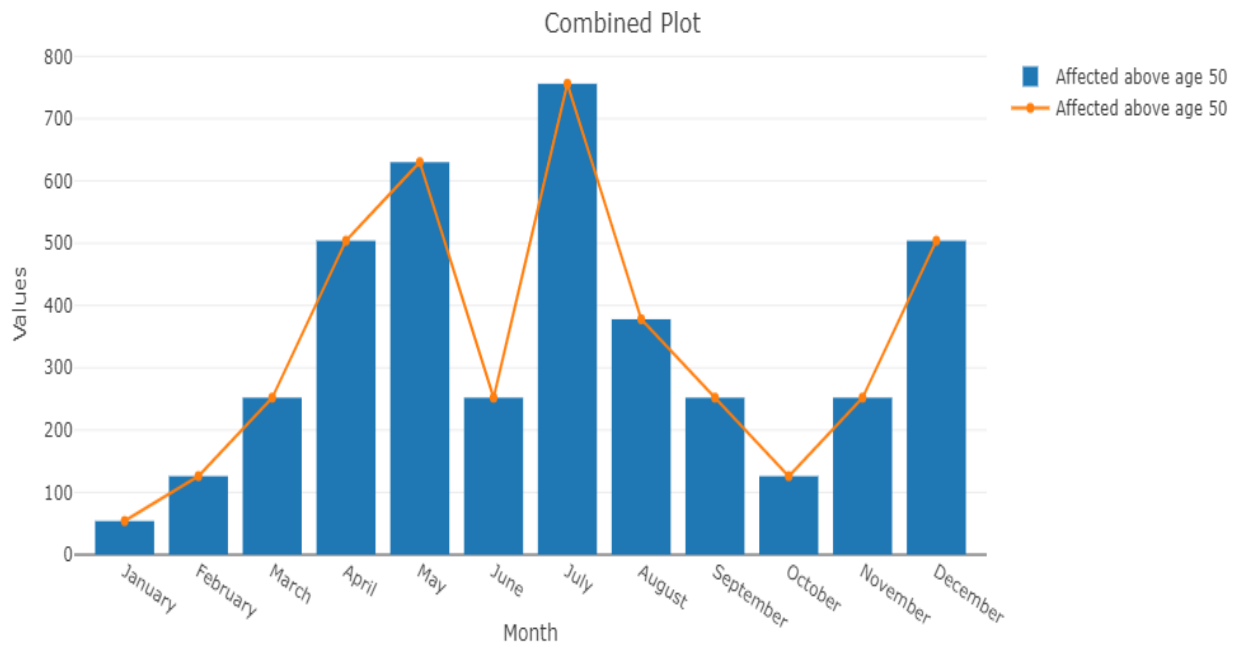
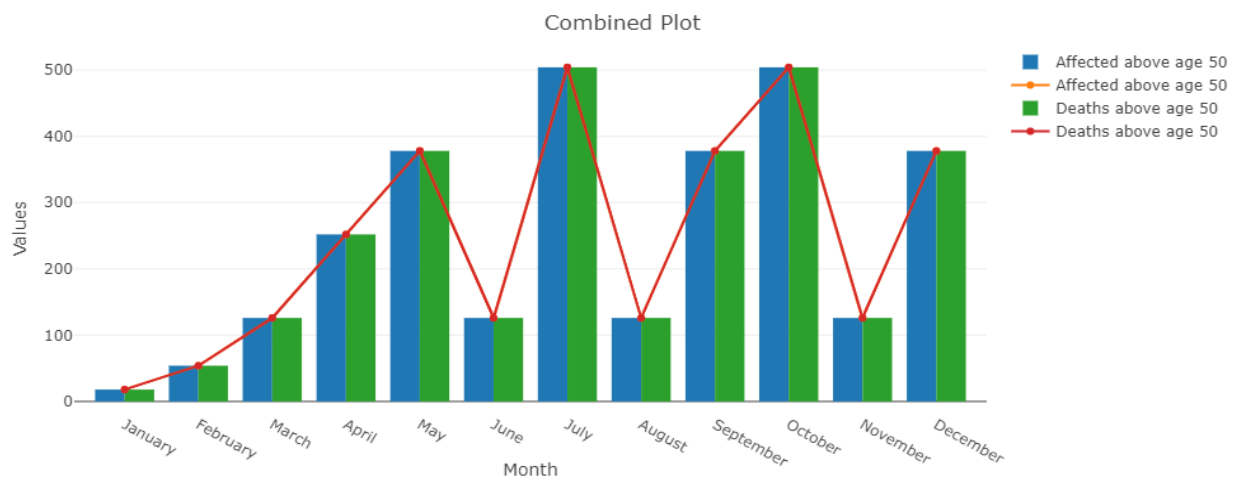
```

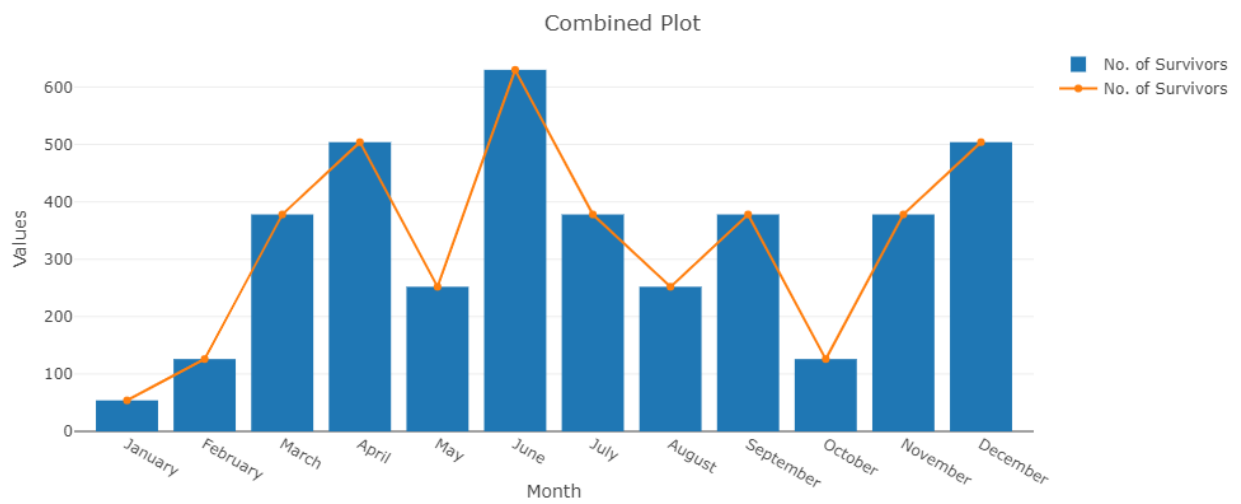
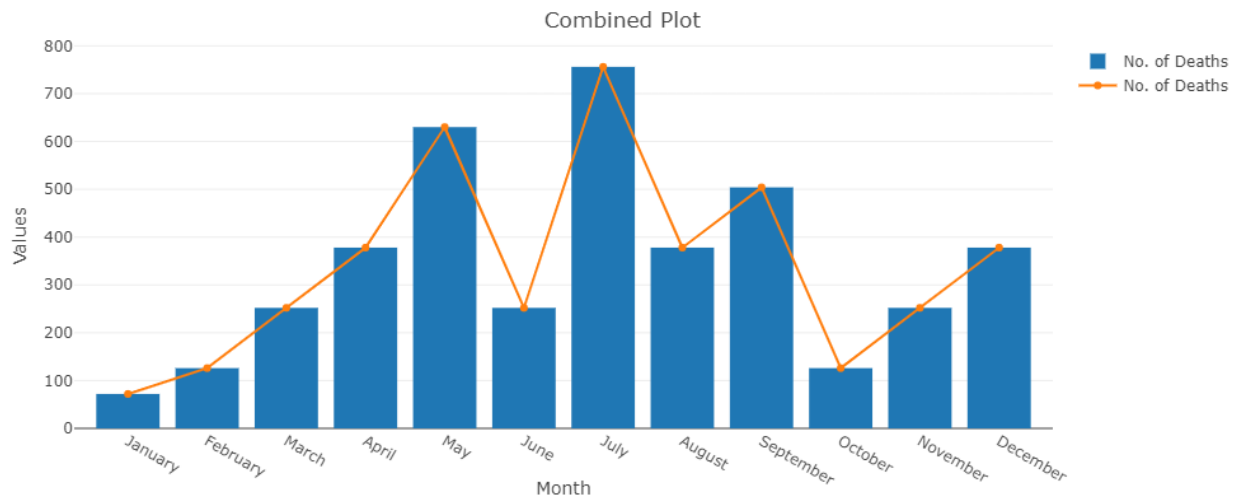
# Create combined plot
output$combined_plot <- renderPlotly({
  req(dataset())
  req(input$variable)

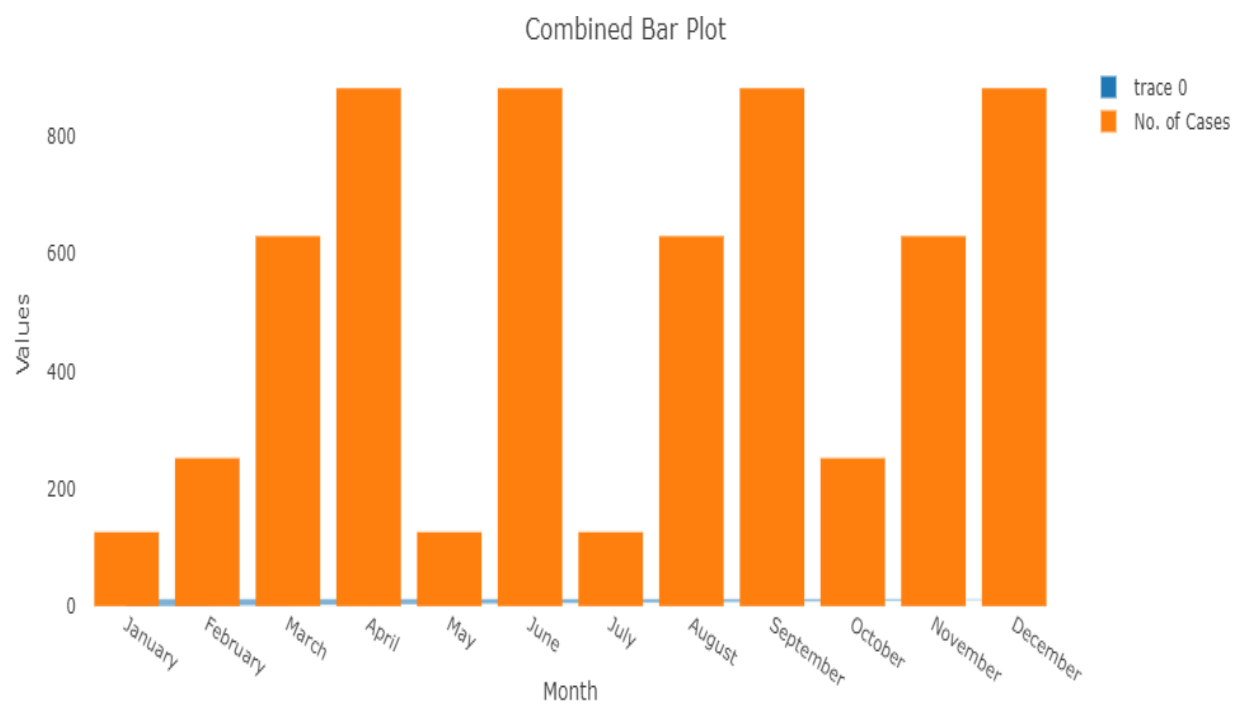
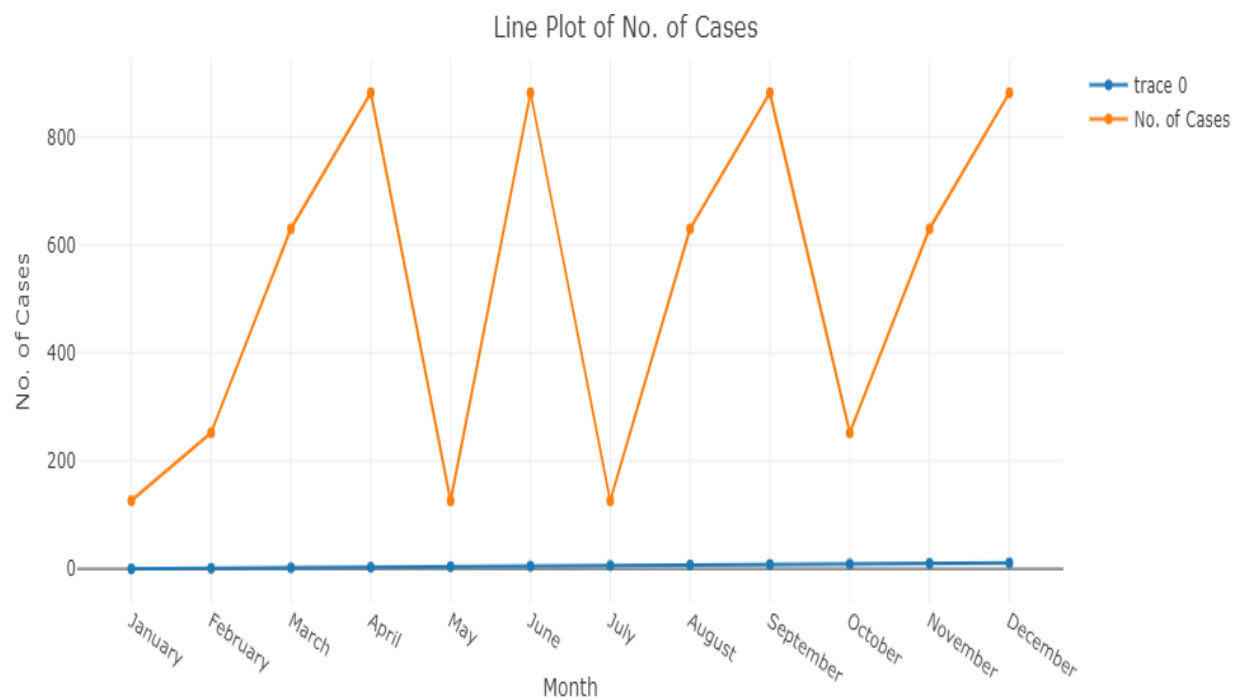
  if (length(input$variable) == 0)
    return(NULL)

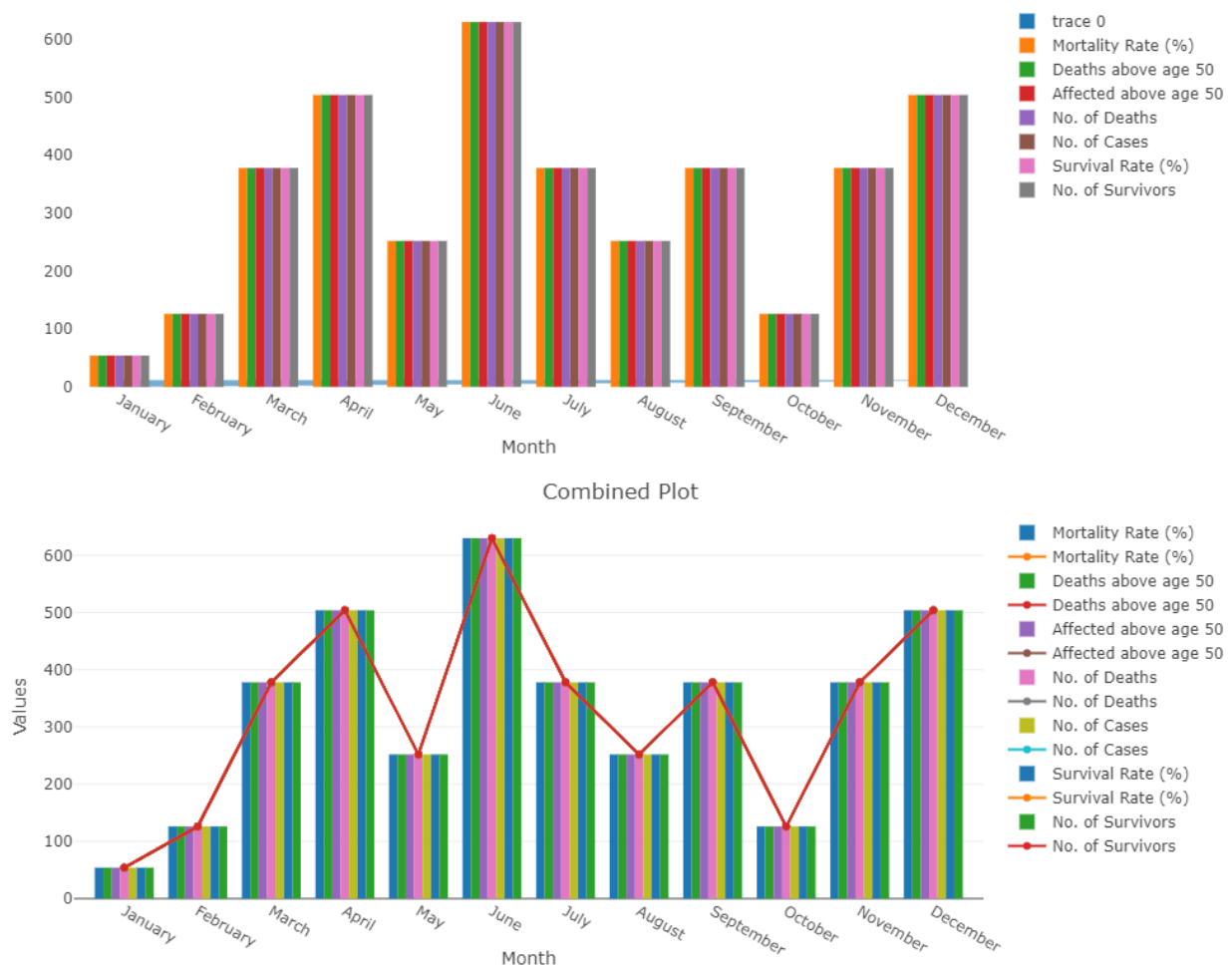
  if (input$plot_type == "Bar Plot") {
    p <- plot_ly(data = dataset(), x = ~Month, type = 'bar') %>%
      layout(title = "Combined Bar Plot",
             xaxis = list(title = "Month"),
             yaxis = list(title = "Values"))
  } else if (input$plot_type == "Line Plot") {
    p <- plot_ly(data = dataset(), x = ~Month, type = 'scatter', mode = 'lines+markers') %>%
      layout(title = "Combined Line Plot",
             xaxis = list(title = "Month"),
             yaxis = list(title = "Values"))
  } else {
    p <- plot_ly(data = dataset(), x = ~Month) %>%
      layout(title = "Combined Plot",
             xaxis = list(title = "Month"),
             yaxis = list(title = "Values"))
  }
})

```









3. User Interface:

- **Intuitive Layout:** Designing an intuitive layout for the Shiny app interface using the fluidPage, titlePanel, sidebarLayout, and mainPanel functions and Organizing the interface elements in a logical manner to guide users through the app's functionality.

- **Input Controls:** Including input controls such as dropdown menus (selectInput) and file upload widgets (fileInput) to allow users to interact with the app and select the desired variables and plot types.

- **Feedback Mechanisms:** Providing feedback mechanisms such as status messages, progress indicators, and error notifications to inform users of the app's current state and any actions they need to take.

4. Interactive Features:

- **Tooltips and Hover Effects:** Using tooltips and hover effects to display additional information when users hover over data points or interact with the plots. This helps users understand the data and interpret the visualizations more effectively.

- **Zooming and Panning:** Enabling zooming and panning functionality to allow users to explore the data in more detail and focus on specific regions of interest within the plots. This enhances the interactivity of the visualizations and enables more in-depth analysis.

```
# UI
ui <- fluidPage(
  titlePanel("COVID-19 Data Visualization"),
  sidebarLayout(
    sidebarPanel(
      fileInput("file", "Choose Excel file (.xlsx format):"),
      selectInput("plot_type", "Select Plot Type:",
        choices = c("Bar Plot", "Line Plot", "Combined")),
      selectInput("variable", "Select Variable:",
        choices = c("No. of Cases", "No. of Deaths", "No. of Survivors",
          "Affected above age 50", "Deaths above age 50",
          "Mortality Rate (%)", "Survival Rate (%)"),
        multiple = TRUE)
    ),
    mainPanel(
      plotlyOutput("barplot"),
      plotlyOutput("lineplot"),
      plotlyOutput("combined_plot"),
      htmlOutput("html_render")
    )
  )
)
```

5. **HTML Rendering:**

- **Dynamic Content:** Use of the `htmlOutput` and `renderUI` functions to render HTML content dynamically within the Shiny app allows for the display of additional information, descriptions, or instructions based on user input or other conditions.

- **Custom Styling:** Applying custom styling to the HTML content using CSS to enhance its visual appearance and ensure consistency with the rest of the app's design and Using CSS classes and inline styles to customize fonts, colors, margins, and other aspects of the HTML elements.

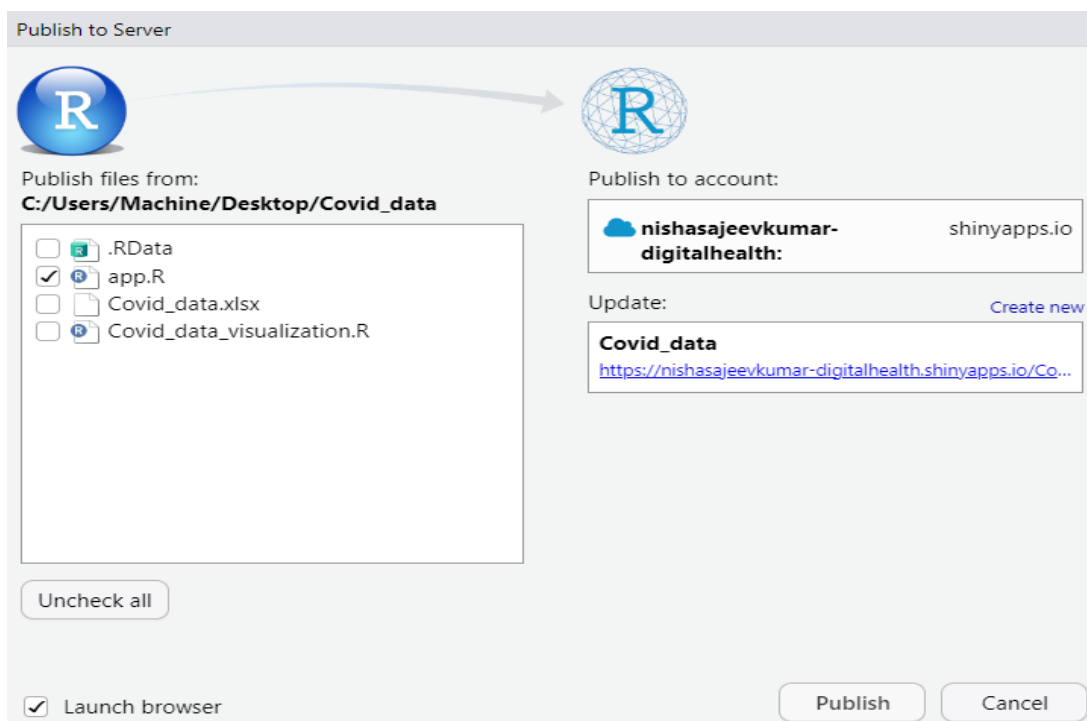
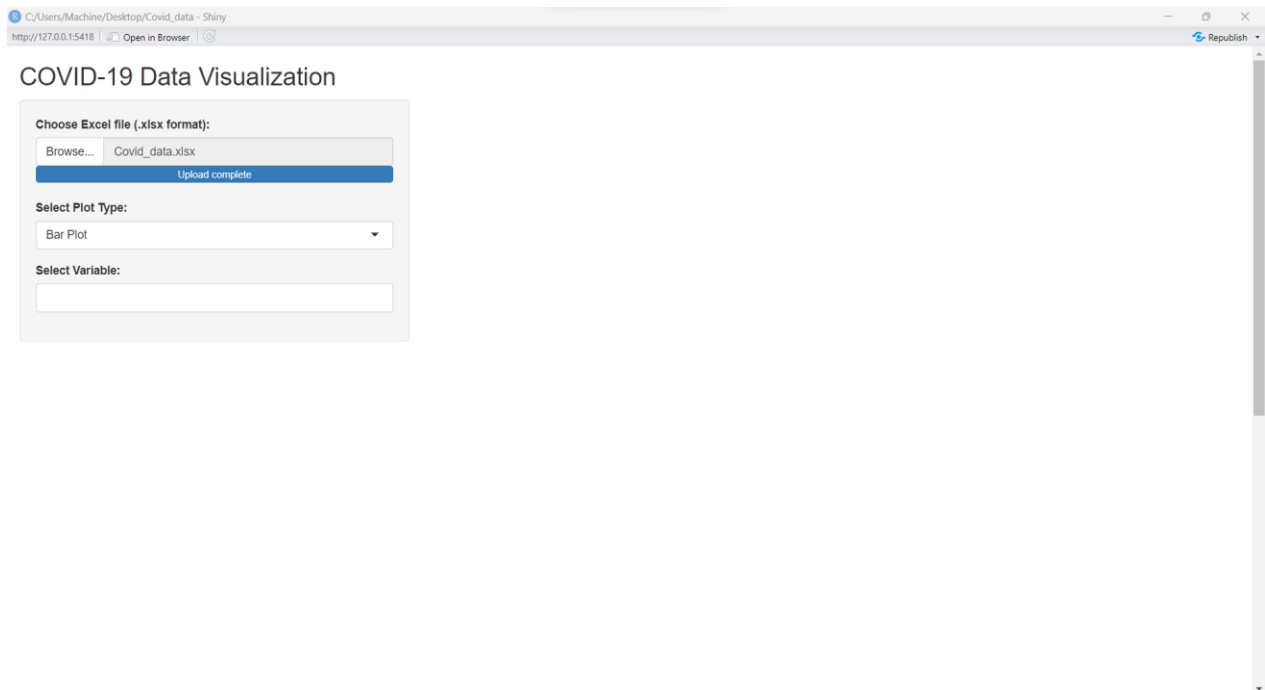
```
# Render HTML content
output$html_render <- renderUI({
  HTML("<h3>NISHA SAJEEV KUMAR</h3>
  <p>Nisha Sajeev Kumar_Digital Health.</p>")
})
```

6. **Deployment:**

- **Hosting Platform:** Choosing a hosting platform such as ShinyApps.io or another cloud service provider deploys the Shiny app. Setting up the necessary account information and authentication tokens to enable deployment from RStudio.

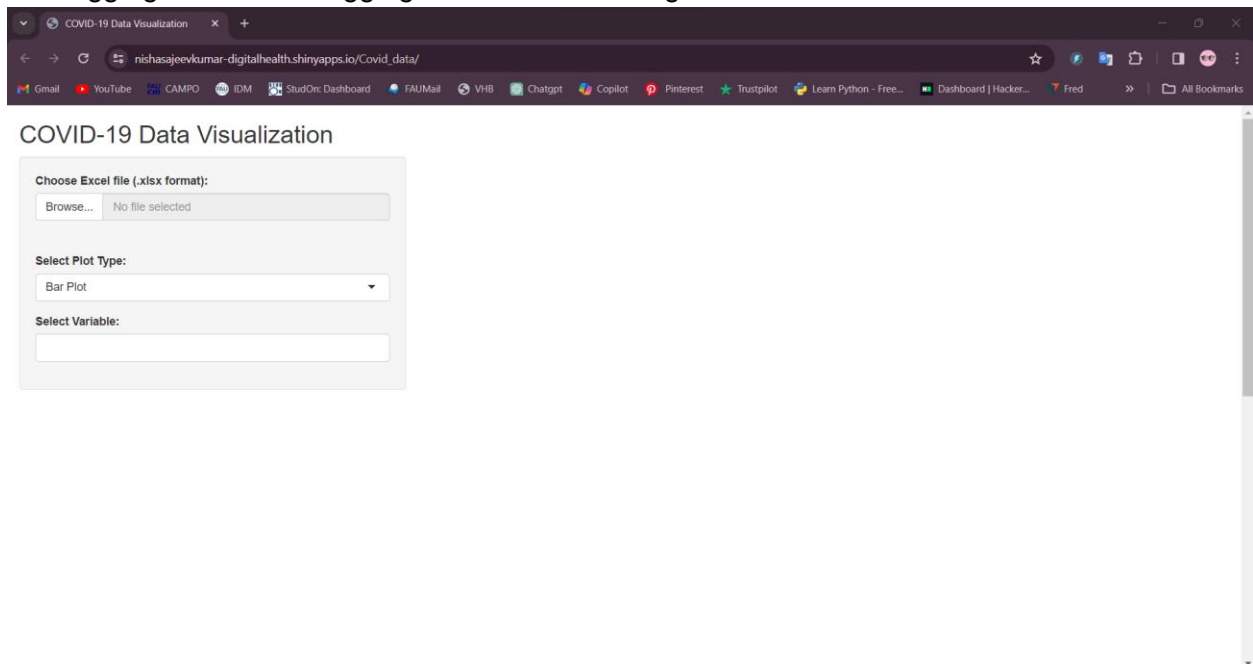
- **Configuration Settings:** Configuring deployment settings such as app title, description, and privacy options. Specifying any required dependencies or packages that need to be installed on the hosting server.

- **Deployment Process:** Using the shinyApp function and the rsconnect package to deploy the Shiny app to the chosen hosting platform and Following the deployment process step-by-step, addressing any issues or errors encountered along the way.



7. **Testing and Debugging:**

- **Functional Testing:** Conducting functional testing to ensure that all features and functionalities of the Shiny app are working as expected. Testing various scenarios, inputs, and user interactions to verify the app's behavior under different conditions.
- **Performance Testing:** Evaluating the performance of the Shiny app, including load times, responsiveness, and scalability, Identifying any bottlenecks or performance issues and optimize the app's code and configuration settings accordingly.
- **Error Handling:** Implementing robust error handling mechanisms to catch and handle any errors or exceptions that may occur during app execution. Providing informative error messages and logging to aid in debugging and troubleshooting.



Browser Window

8. **Documentation and Maintenance:**

- **User Guide:** Creating a comprehensive user guide or documentation for the Shiny app, including instructions on how to use the app, interpret the visualizations, and troubleshoot common issues. Making the documentation easily accessible within the app or on a dedicated website.
- **Version Control:** Use of version control systems such as Git to track changes to the app's codebase and collaborate with other developers or contributors. Maintaining clear commit messages and documentation to facilitate code review and future updates.
- **Regular Updates:** Regularly updating the Shiny app with new features, improvements, and updated data to keep it relevant and effective. Solicit feedback from users and stakeholders can identify areas for enhancement and prioritize future development efforts accordingly.

By following these detailed steps and settings in the tools used, the dynamic data visualization tool is effectively developed and deployed to support stakeholders in addressing the challenges of the COVID-19 pandemic.

Journey:

The journey from the initial prototype to the final version of the Shiny app represents a process of iterative development, experimentation, and refinement. Here's a detailed elaboration of this journey:

1. Initial Prototype Development:

- The journey begins with the creation of an initial prototype of the Shiny app. This prototype contains basic functionality, such as data input, simple plot generation, and a rudimentary user interface.
- During this stage, the focus is on establishing the core features and functionality of the app, without delving into advanced visualizations or interactive elements.

2. Exploration of Plot Types and Layouts:

- As development progresses, we explore different plot types, layouts, and visualizations to determine the most effective ways to present the COVID-19 data.
- Experimenting with various plot types, including bar plots, line plots, scatter plots, and heatmaps, helps identify which visualization techniques best convey the trends and patterns in the data.
- Layout considerations involve determining the optimal arrangement of plots, input controls, and other interface elements to create a cohesive and intuitive user experience.

3. Integration of Interactive Features:

- Integrating interactive features into the app to enhance user engagement and exploration of the data. This includes adding tooltips, hover effects, zooming, panning, and selection filters to the plots.
- Interactive elements empower us to interact directly with the visualizations, allowing us to drill down into specific data points, compare trends across variables, and gain deeper insights into the data.

4. Challenges in Data Cleaning and Optimization:

- Throughout the development process, there were challenges related to data cleaning, transformation, and optimization.
- Data cleaning involved identifying and addressing inconsistencies, missing values, outliers, and other data quality issues in the COVID-19 dataset. To make sure that the visualizations accurately reflect the underlying data.
- Performance optimization efforts focus on improving the speed and efficiency of the app, particularly as the size and complexity of the dataset grow.

5. Iterative Development and Refinement:

- The development process follows an iterative approach, with multiple cycles of testing, feedback, and refinement.
- Each iteration builds upon the previous one, incorporating lessons learned, addressing identified issues, and introducing new features and enhancements.

6. Leveraging R Shiny and Plotly Capabilities:

- Throughout the journey, R Shiny and plotly have been used to develop a dynamic and interactive data visualization tool.
- R Shiny provides a powerful framework for building web applications with R, allowing for the creation of responsive, user-friendly interfaces.
- Plotly offers a rich set of visualization tools and features, enabling the creation of interactive plots and dashboards that facilitate data exploration and analysis.

A systematic approach and embracing the iterative nature of software development, navigates the journey from the first prototype to the final version of the Shiny app. Along the way, challenges overcome and harnessed the capabilities of R Shiny and plotly to develop a valuable resource for addressing the challenges of the COVID-19 pandemic.