UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

PROJECT REPORT ON

STOCK MARKET ANALSER

UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

MASTERS OF COMPUTER APPLICATIONS

[AIML]

STATISTICAL TECHNIQUES USING R LAB

SUBMITTED BY :-	SUBMITTED TO:
SNEHA	MS. MAUSAM
UID : 24MCI10090	E17645
SUBJECT CODE : 24CAP-612	ASSISTANT PROFESSOR

1. Aim:

To create an interactive web application that allows users to visualize stock price trends and analyze performance based on selected stocks and date ranges.

2. Task to be done:

Steps to Follow:

1. Set Up Your R Environment:

- Install and load the required packages:

Code: install.packages(c("shiny", "quantmod", "ggplot2", "dplyr", "lubridate"))

```
        library(shiny)
library(quantmod)
library(ggplot2)
library(dplyr)
library(lubridate)
```

2. Create the UI (User Interface):

- Define the layout of your app. The UI will include input fields for selecting a stock symbol and a date range for analysis.

Code : ui <- fluidPage(

```
        titlePanel("Interactive Stock Market Analyzer"),
        idebarLayout(
        sidebarPanel(
        textInput("stockSymbol", "Enter Stock Symbol (e.g., AAPL, GOOGL):", value =
          "AAPL"),
dateInput("startDate", "Select Start Date:", value = "2022-01-01"),
dateInput("endDate", "Select End Date:", value = Sys.Date()),
actionButton("analyze", "Analyze Stock")
 ),
 mainPanel(
plotOutput("stockPlot"),
verbatimTextOutput("summary")
 )
 )
 )
```

3. Create the Server Function:

- This function will handle the logic of your application, including retrieving stock data and generating plots.

Code : server <- function(input, output) {

```
observeEvent(input$analyze, {
 req(input$stockSymbol)  # Ensure that a stock symbol is provided

 # Retrieve stock data

stock_data <- tryCatch({
getSymbols(input$stockSymbol, src = "yahoo", auto.assign = FALSE,
from = input$startDate, to = input$endDate)
 }, error = function(e) {
return(NULL)
  })

 if (is.null(stock_data)) {
```

```
 output$summary <- renderPrint("Error: Stock data could not be retrieved.
Please check the stock symbol.")
 return()
 }

# Prepare the data for plotting
 stock_data <- data.frame(Date = index(stock_data), coredata(stock_data))
colnames(stock_data) <- c("Date", "Open", "High", "Low", "Close", "Volume",
"Adjusted")

 # Plot the stock price
 output$stockPlot <- renderPlot({
 ggplot(stock_data, aes(x = Date, y = Close)) +
geom_line(color = "blue") +
labs(title = paste("Stock Price for", input$stockSymbol),
x = "Date", y = "Closing Price (USD)") +
theme_minimal()
 })

# Display summary statistics
output$summary <- renderPrint({
summary(stock_data$Close)
 })
 })
  }
```

4. Run the Application:

- Combine the UI and server functions to run the Shiny app:

Code : shinyApp(ui = ui, server = server)

5. Test the App:

- Run your R script. The Shiny app should open in a web browser where you can enter a stock symbol and a date range to analyze stock prices. You will see the stock price trend plotted along with summary statistics.

3. Libraries And Languages Used:

- **R Programming Language:**

o The core language for data analysis and building the web application.

- **Libraries used:**

o shiny: For creating the interactive web application.
o quantmod: To retrieve stock market data from financial APIs (e.g., Yahoo Finance).
o ggplot2: For visualizing stock trends.
o dplyr: For data manipulation and transformation.
o lubridate: To handle date operations efficiently.

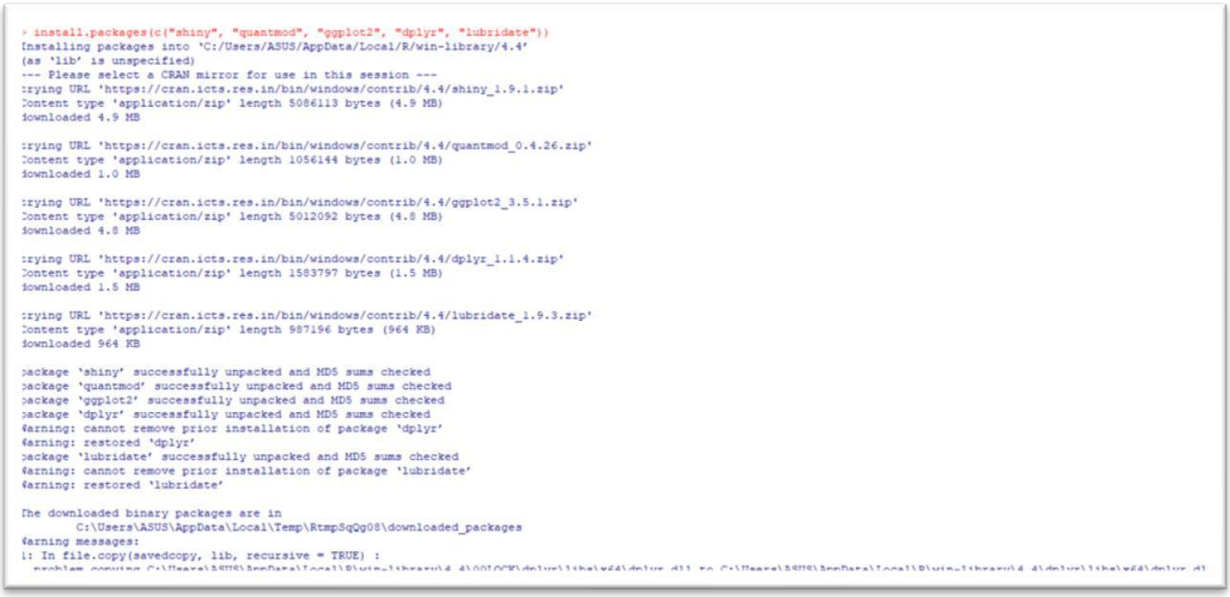
- **User Interface:**

o textInput: A field for users to enter a stock symbol.
o dateInput: Input fields to select the date range for the analysis.
o actionButton: A button to trigger the analysis.
o plotOutput: Displays the stock price trend plot. o verbatimTextOutput: Displays summary statistics for the stock prices.

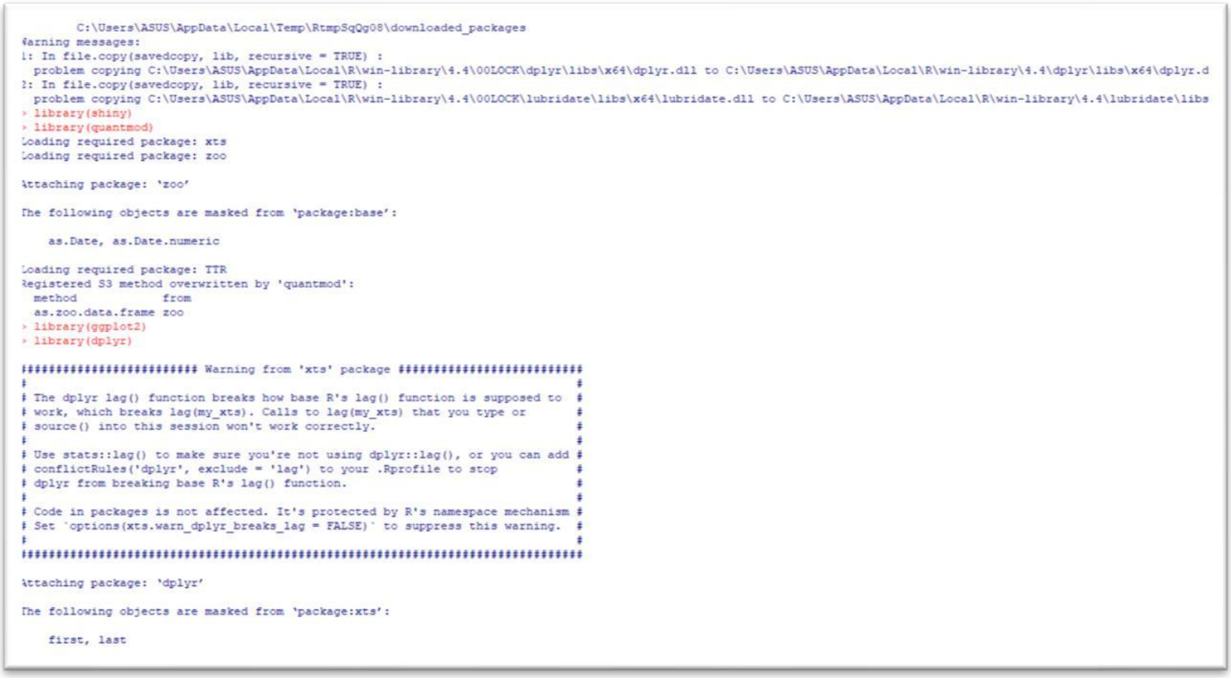
- **Server Logic:**

o The server retrieves stock data from Yahoo Finance using getSymbols() when the "Analyze Stock" button is clicked.
o The data is prepared for plotting, and a line plot shows the closing price over the selected date range.
o Summary statistics for the closing price are also displayed.

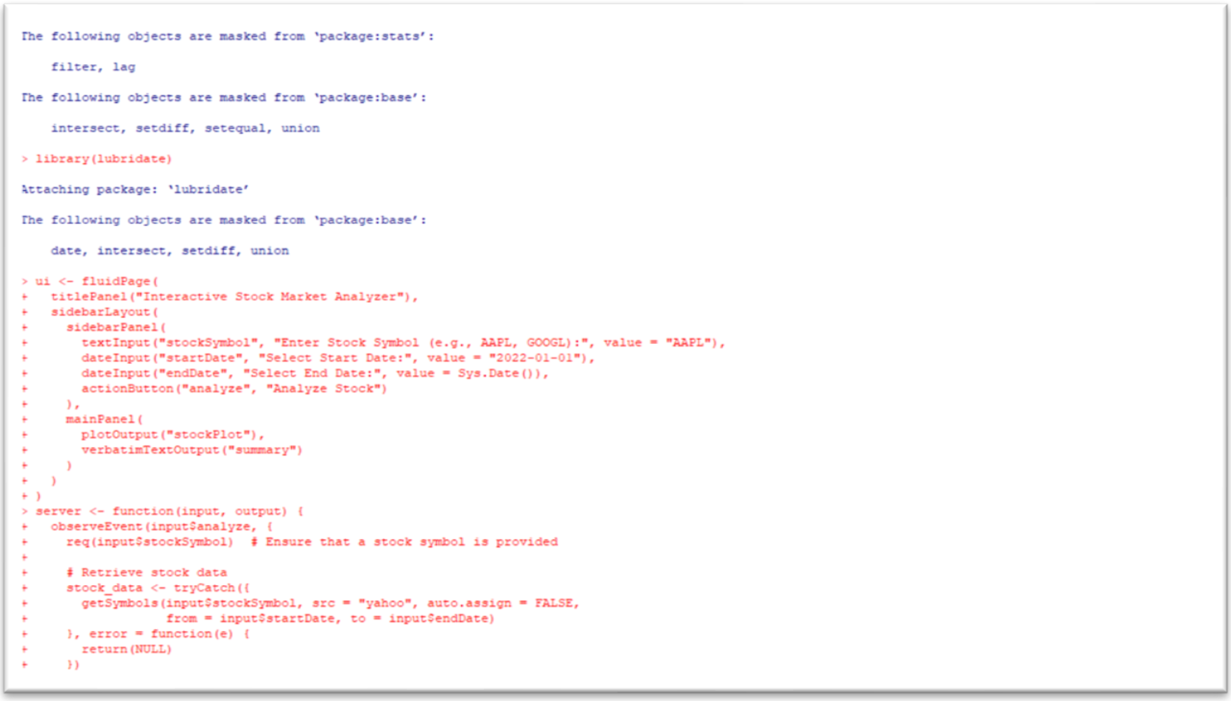
4. Implementation:



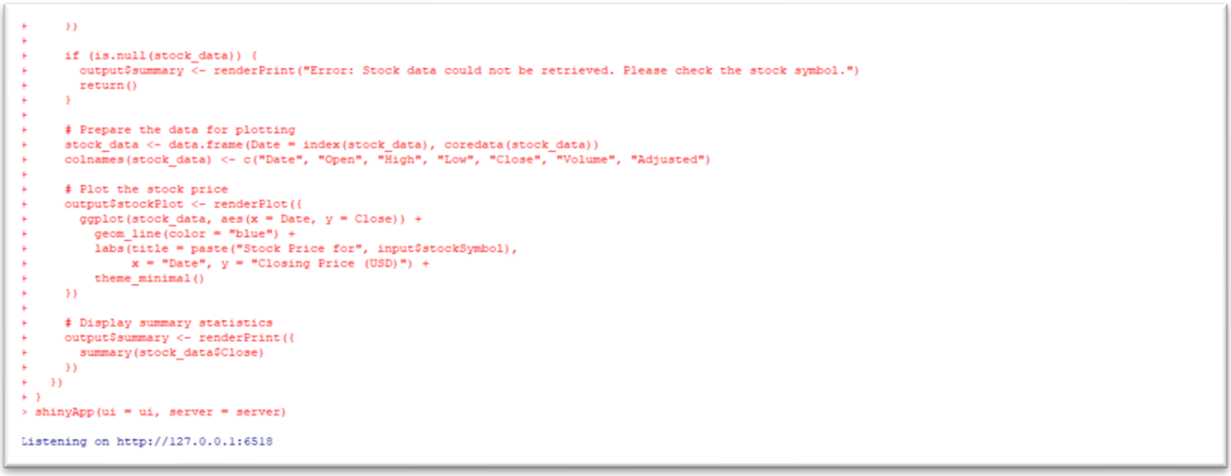
```
> install.packages(c("shiny", "quantmod", "ggplot2", "dplyr", "lubridate"))
Installing packages into 'C:/Users/ASUS/AppData/Local/R/win-library/4.4'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cran.icts.res.in/bin/windows/contrib/4.4/shiny_1.9.1.zip'
Content type 'application/zip' length 5086113 bytes (4.9 MB)
downloaded 4.9 MB

trying URL 'https://cran.icts.res.in/bin/windows/contrib/4.4/quantmod_0.4.26.zip'
Content type 'application/zip' length 1056144 bytes (1.0 MB)
downloaded 1.0 MB

trying URL 'https://cran.icts.res.in/bin/windows/contrib/4.4/ggplot2_3.5.1.zip'
Content type 'application/zip' length 5012092 bytes (4.8 MB)
downloaded 4.8 MB

trying URL 'https://cran.icts.res.in/bin/windows/contrib/4.4/dplyr_1.1.4.zip'
Content type 'application/zip' length 1583797 bytes (1.5 MB)
downloaded 1.5 MB

trying URL 'https://cran.icts.res.in/bin/windows/contrib/4.4/lubridate_1.9.3.zip'
Content type 'application/zip' length 987196 bytes (964 KB)
downloaded 964 KB

package 'shiny' successfully unpacked and MD5 sums checked
package 'quantmod' successfully unpacked and MD5 sums checked
package 'ggplot2' successfully unpacked and MD5 sums checked
package 'dplyr' successfully unpacked and MD5 sums checked
Warning: cannot remove prior installation of package 'dplyr'
Warning: restored 'dplyr'
package 'lubridate' successfully unpacked and MD5 sums checked
Warning: cannot remove prior installation of package 'lubridate'
Warning: restored 'lubridate'

The downloaded binary packages are in
        C:\Users\ASUS\AppData\Local\Temp\RtmpSqQg08\downloaded_packages
Warning messages:
1: In file.copy(savedcopy, lib, recursive = TRUE) :
  problem copying C:\Users\ASUS\AppData\Local\R\win-library\4.4\00LOCK\dplyr\libs\x64\dplyr.dll to C:\Users\ASUS\AppData\Local\R\win-library\4.4\dplyr\libs\x64\dplyr.dl
```



```
        C:\Users\ASUS\AppData\Local\Temp\RtmpSqQg08\downloaded_packages
Warning messages:
1: In file.copy(savedcopy, lib, recursive = TRUE) :
  problem copying C:\Users\ASUS\AppData\Local\R\win-library\4.4\00LOCK\dplyr\libs\x64\dplyr.dll to C:\Users\ASUS\AppData\Local\R\win-library\4.4\dplyr\libs\x64\dplyr.d
2: In file.copy(savedcopy, lib, recursive = TRUE) :
  problem copying C:\Users\ASUS\AppData\Local\R\win-library\4.4\00LOCK\lubridate\libs\x64\lubridate.dll to C:\Users\ASUS\AppData\Local\R\win-library\4.4\lubridate\libs
> library(shiny)
> library(quantmod)
Loading required package: xts
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric

Loading required package: TTR
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
> library(ggplot2)
> library(dplyr)

######################### Warning from 'xts' package ##########################
#                                                                             #
# The dplyr lag() function breaks how base R's lag() function is supposed to  #
# work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
# source() into this session won't work correctly.                            #
#                                                                             #
# Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
# conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
# dplyr from breaking base R's lag() function.                                #
#                                                                             #
# Code in packages is not affected. It's protected by R's namespace mechanism #
# Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
#                                                                             #
###############################################################################

Attaching package: 'dplyr'

The following objects are masked from 'package:xts':

    first, last
```

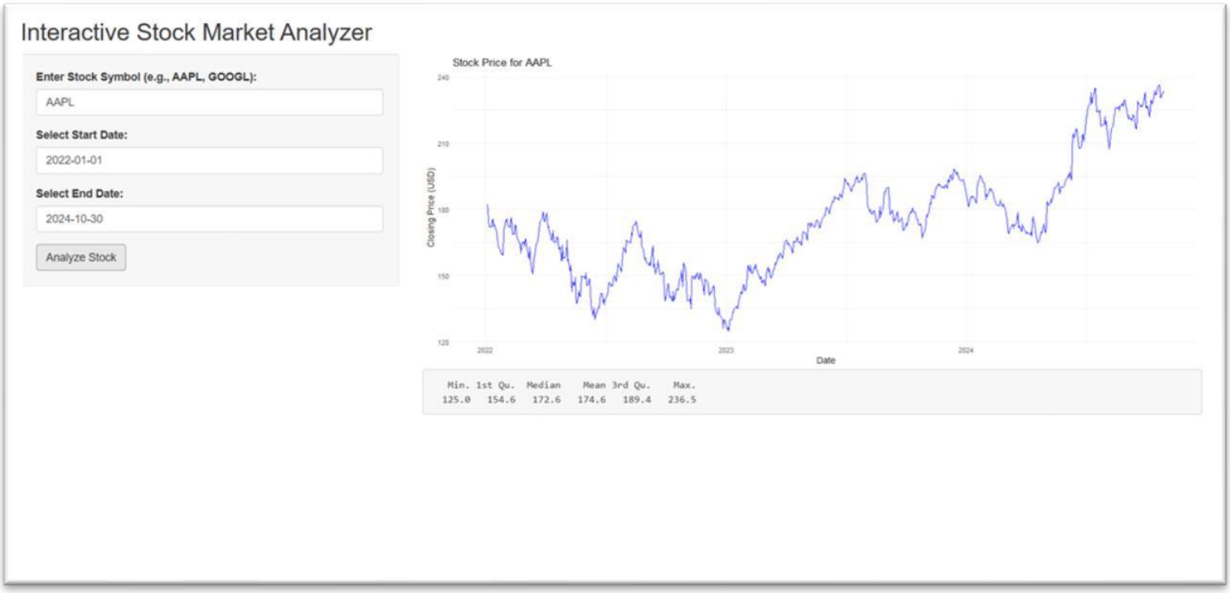


```
The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> library(lubridate)

Attaching package: ‘lubridate’

The following objects are masked from ‘package:base’:

    date, intersect, setdiff, union

> ui <- fluidPage(
+   titlePanel("Interactive Stock Market Analyzer"),
+   sidebarLayout(
+     sidebarPanel(
+       textInput("stockSymbol", "Enter Stock Symbol (e.g., AAPL, GOOGL):", value = "AAPL"),
+       dateInput("startDate", "Select Start Date:", value = "2022-01-01"),
+       dateInput("endDate", "Select End Date:", value = Sys.Date()),
+       actionButton("analyze", "Analyze Stock")
+     ),
+     mainPanel(
+       plotOutput("stockPlot"),
+       verbatimTextOutput("summary")
+     )
+   )
+ )
> server <- function(input, output) {
+   observeEvent(input$analyze, {
+     req(input$stockSymbol)  # Ensure that a stock symbol is provided
+
+     # Retrieve stock data
+     stock_data <- tryCatch({
+       getSymbols(input$stockSymbol, src = "yahoo", auto.assign = FALSE,
+                  from = input$startDate, to = input$endDate)
+     }, error = function(e) {
+       return(NULL)
+     })
```



```
+     })
+
+     if (is.null(stock_data)) {
+       output$summary <- renderPrint("Error: Stock data could not be retrieved. Please check the stock symbol.")
+       return()
+     }
+
+     # Prepare the data for plotting
+     stock_data <- data.frame(Date = index(stock_data), coredata(stock_data))
+     colnames(stock_data) <- c("Date", "Open", "High", "Low", "Close", "Volume", "Adjusted")
+
+     # Plot the stock price
+     output$stockPlot <- renderPlot({
+       ggplot(stock_data, aes(x = Date, y = Close)) +
+         geom_line(color = "blue") +
+         labs(title = paste("Stock Price for", input$stockSymbol),
+              x = "Date", y = "Closing Price (USD)") +
+         theme_minimal()
+     })
+
+     # Display summary statistics
+     output$summary <- renderPrint({
+       summary(stock_data$Close)
+     })
+   })
+ }
> shinyApp(ui = ui, server = server)

Listening on http://127.0.0.1:6518
```

5. Output:



Interactive Stock Market Analyzer

Enter Stock Symbol (e.g., AAPL, GOOGL): AAPL

Select Start Date: 2022-01-01

Select End Date: 2024-10-30

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 125.0   154.6   172.6   174.6   189.4   236.5 
```

6. Conclusions:

The Interactive Stock Market Analyzer project demonstrates how R Shiny can be utilized to build user-friendly applications that combine financial data analysis with visualization. The project achieves key goals in enabling users to:

1. Visualize Stock Trends:

- Users can generate customized time-series plots of stock prices, gaining insights into performance over specific periods.

2. Perform Basic Analysis:

- By providing summary statistics (e.g., minimum, maximum, median), the tool supports quick analysis of stock behavior.

3. User Engagement and Ease of Use:

- With a simple interface to input stock symbols and select date ranges, even novice users can interact with stock data effortlessly.

4. Dynamic Data Handling:

- The use of quantmod to fetch real-time stock data from Yahoo Finance ensures that the app stays current and eliminates the need for manual data downloads.

This project highlights the potential of R Shiny in building interactive financial dashboards, demonstrating both data retrieval and visualization capabilities.

7. Future Frameworks and Enhancements:

1) Add Technical Indicators and Moving Averages:

- Enhance the application by incorporating technical indicators such as:
  - 50-day and 200-day Moving Averages (SMA).
  - Exponential Moving Average (EMA).
  - MACD (Moving Average Convergence Divergence).
- Users can visualize these indicators alongside stock price trends to better understand market behavior.

Code : stock_data$SMA50 <- SMA(stock_data$Close, n = 50)
stock_data$SMA200 <- SMA(stock_data$Close, n = 200)

2) Comparison of Multiple Stocks:

- Enable users to analyze and compare multiple stock symbols on a single plot.
- This will require checkbox inputs or selectInput() elements to allow for multi-stock selection.

Code : selectInput("stocks", "Choose Stocks", choices = c("AAPL", "GOOGL", "MSFT"), multiple = TRUE)

3) Enhanced Data Visualization:

- Add candlestick charts to provide detailed insights into the market's daily open, high, low, and close prices.
- Use the plotly library to make charts interactive, allowing users to zoom, hover, and explore data points.

4) Export and Download Options:

- Provide an option for users to download the retrieved stock data as a CSV file.
- This would allow users to perform further offline analysis or store the data.

Code : downloadHandler(

```
        filename = function() { paste(input$stockSymbol, "data.csv", sep = "_") },
content = function(file) { write.csv(stock_data, file) }
        )
```

5) Error Handling and Validation:

- Improve error handling by validating invalid or unavailable stock symbols with better user feedback.

- Add a loading indicator or progress bar to improve the user experience while data is being fetched.

6) Real-Time Data Integration:

- Integrate with financial APIs like Alpha Vantage or IEX Cloud to fetch live market data instead of relying solely on Yahoo Finance for historical data.

7) Mobile-Friendly UI and Theme Enhancements:

- Use Shiny themes or integrate with Bootstrap CSS to create a responsive layout that works on mobile devices.

- Enhance the appearance using shinyWidgets for an improved user interface.

8) Deployment to the Cloud:

- Deploy the application on shinyapps.io or RStudio Connect to make the tool publicly accessible and shareable.

- Optionally, integrate with Google Analytics to track user engagement and performance of the application.

8. <u>Learning Outcomes:</u>

1. Hands-On Experience with R and Shiny:

- You will gain practical knowledge of Shiny, a web framework for building interactive applications using R.
- Learn how to build dynamic web UIs with inputs (e.g., text, date selectors) and outputs (e.g., plots, tables).

2. Data Retrieval and Financial Data Analysis:

- Understand how to retrieve financial data from Yahoo Finance using the quantmod package.
- Gain insights into handling stock market data, including Open, High, Low, Close, and Adjusted prices.

3. Visualization Skills with ggplot2:

- Learn to use ggplot2 to create time-series plots, visualize stock trends, and customize charts with titles, labels, and themes.
- Understand how to create line plots, explore the use of color themes, and experiment with axes labeling for financial data.

4. Data Manipulation and Summarization with dplyr:

- Learn basic data manipulation with dplyr and understand how to transform raw data into meaningful formats.
- Explore how to generate summary statistics (mean, median, min, max) for stock prices to aid in performance analysis.

5. User Interaction and Event Handling:

- Learn to use reactive programming concepts with Shiny (e.g., observeEvent()) to trigger actions based on user inputs.
- Understand how to manage user interactions such as symbol input, date range selection, and button events to update visualizations dynamically.

6. Error Handling and Data Validation:

- Develop skills in handling errors gracefully using tryCatch() to manage situations when stock data is unavailable or symbols are incorrect.
- Understand how to provide user feedback with appropriate messages for errors and missing data.

7. Working with Dates and Time-Series Data:

- Learn to manage time-series data and filter stock prices based on user-selected date ranges using the lubridate package.
- Understand the importance of date formats in analyzing financial trends.

8. Building and Deploying Web Applications:

- Gain experience in creating a full-fledged web application by integrating UI and server logic using shinyApp().
- Understand how to run and test Shiny applications locally and learn how to troubleshoot common issues.

9. Future Scalability and Enhancements:

- Understand the potential to scale the project with advanced features, including:

○ **Technical indicators** (e.g., moving averages, MACD).
○ **Comparing multiple stocks** on a single plot.
○ **Interactive charts** using plotly. ○ **Data export functionality** for further analysis in external tools.

10. Introduction to Financial Analysis Concepts:

- Become familiar with **basic financial analysis** by interpreting stock price trends and summary statistics.
- Understand how **closing prices** reflect market sentiment and how technical indicators assist in decision-making.