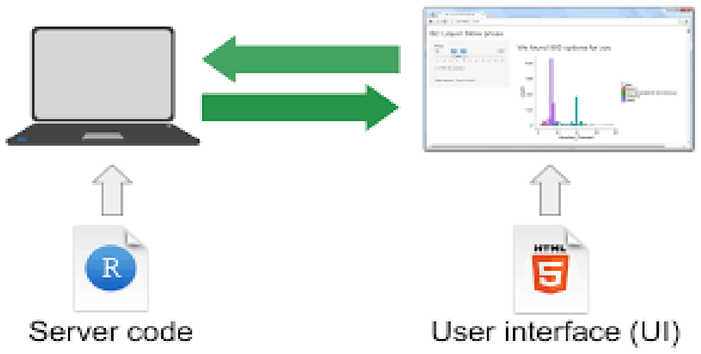
Web Application for Statistical Analysis



PREFACE

A practical knowledge in a student's life is very important. It helps a student to know the real-life situation and problems of life. Theoretical knowledge is very much needed but practical knowledge is equally important. This practical knowledge to a student is given in a form of project.

This project provides an opportunity and platform to know the current situation and the behaviour of environment.

For the preparation of this project, I feel deep sense of gratitude to all faculty members, staff members and Internship members of the respective organization and all other persons who helped me to prepare such project report.

CONTENTS

LIST OF FIGURES

List of Table

ABSTRACT

As statistics grows in importance in our society because of the necessity to analyse the increasing amounts of knowledge available, it's necessary to strengthen the role of statistics in education. However, the non-existence of free open-source statistical tools designed specifically for educational purposes make the task harder. Concern about this fact led to the present project. This project will introduce web application which is an interactive and R-Shiny based application that may perform popular statistical analysis that are frequently used. The applying can perform even the easy rectilinear regression, Correlation etc., together with the summary plots for users with no prior R programming or perhaps with limited statistical knowledge. This application also eliminates the requirement for various lines of programming effort to make an identical statistical analysis.

1. INTRODUCTION

Shiny is an R package that produces it easy to create interactive web apps straight from R. This webinar will cover the fundamentals of building and deploying a Shiny app, including essentials of reactive programming for building apps that are more efficient, robust, and proper. We'll also discuss building interactive plots and creating interactive documents containing Shiny components. Shiny makes it incredibly easy to make interactive web applications with R. Automatic "reactive" binding between inputs and outputs and extensive prebuilt widgets make it possible to make beautiful, responsive, and powerful applications with minimal effort.

R Shiny = R + interactivity + web made easy

1.1 Features

- Build useful web applications with just some lines of code—no JavaScript required.
- Shiny applications are automatically "live" within the identical way that spreadsheets are live. Outputs change instantly as users modify inputs, without requiring a reload of the browser.
- Shiny user interfaces is also built entirely using R, or is written directly in HTML, CSS, and JavaScript for more flexibility.
- Works in any R environment (Console R, Rgui for Windows or Mac, ESS, StatET, RStudio, etc.)
- Attractive default UI theme supported Twitter Bootstrap.
- Pre-built output widgets for displaying plots, tables, and printed output of R objects.
- Uses a reactive programming model that eliminates messy event handling code, so you will be ready to focus on the code that truly matters

1.2 Installation:

Shiny is available on CRAN, so you can install it in the usual way from your R console:

```
install.packages("shiny")
```

2. OBJECTIVE

The project tries to face the difficulties in the design and implementation of an app thought learn statistics and data analysis. The main objectives of the project are the following:

1. To perform an exhaustive research of already existing statistical software (including both commercial and open-source programs, as well as programming languages).
2. This is very important to understand what already exists, and to try to find what is needed. It also helps to get inspired towards the design of application.
3. To decide to whom the app is addressed (users) and exhaustively analysing their Needs.
4. To define the list of characteristics of the program.
5. To design theoretically all the features and functionalities of application, as well as its User Interface.
6. To select a computational tool to build the app.
7. To build a prototype of application, using the selected computational tool. More specifically, building the inner architecture of the app, and implementing the functionalities, always keeping in mind to make application easily expandable.
8. To test and validate the design of the app, using design techniques.
9. Reduce coding time and Convenient for analysis. Fast processing and analysis

3. STRUCTURE OF R-SHINY

3.1 What is R Shiny?

Shiny is an R package that's capable of building an interactive online page application straight from R without using any web application languages like HTML, CSS, or JavaScript knowledge.

One essential feature of Shiny is that these applications are in a very way “live” since the output of the online page changes because the user modifies the inputs, without reloading the browser.

Shiny contains two fundamental parameters, the UI and also the Server. The computer programme (UI) holds all the text code that describes the layout of the page, any additional text, images, and other HTML elements we would like to incorporate therefore the user can interact and understand a way to use the net page. On the opposite hand, the Server is that the backside of the Shiny application. This parameter creates an online server specifically designed to host Shiny apps in a very controlled environment

3.2 The basic parts of a Shiny app

Shiny applications have two components, a program object and a server function, that are passed as arguments to the shinyApp function that makes a Shiny app object from this UI/server pair. The ASCII text file for both of those components is listed below. In subsequent sections of the article, we’ll break down Shiny code thoroughly and explain the employment of “reactive” expressions for generating output. For now, just try wiggling with the sample application and reviewing the ASCII text file to urge an initial condole with things. Make certain to read the comments carefully.

3.3 How to launch a Shiny app

In previous you’ve been calling runApp to run the instance applications. This function starts the applying and reveal your default applications programme to look at it. The decision is obstructing, meaning that it prevents traditional interaction with the console while the applying is running. To prevent the applying you merely interrupt R – you'll do that by pressing the Ctrl-C in some R front ends, or the Escape key in RStudio, or by clicking the stop button if your R environment provides one.

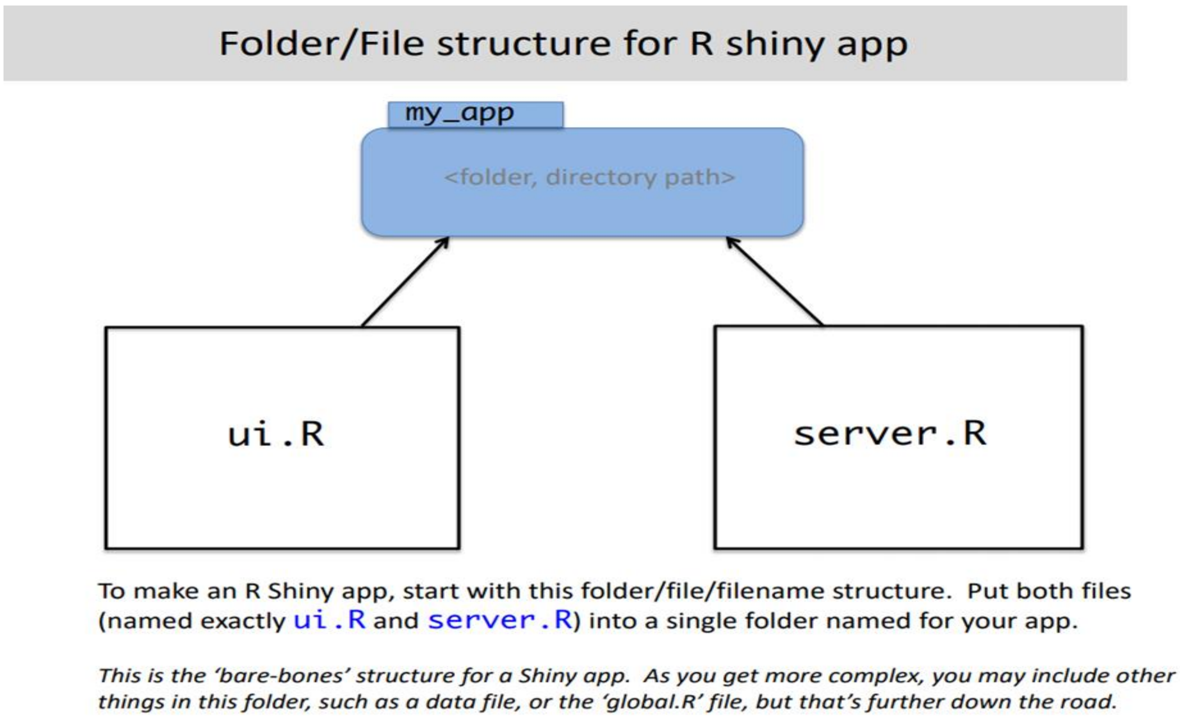
3.3.1 Shiny applications have two components:

– a user-interface definition (UI) file called ui.R

This source code is used to set-up what the user will actually see in the web app, i.e. the layout of the web page – Title, sliders, widgets, plots, location of items on the page, etc. • This source code is also used to accept input from the user – e.g. It recognizes what the user has entered in the slider.

– a server script file called server.R

This source code does the computational R work "under the hood" with familiar functions such as hist(), plot(), etc. • This source code contains the instructions that your computer needs to build your app.



To make an R Shiny app, start with this folder/file/filename structure. Put both files (named exactly ui.R and server.R) into a single folder named for your app.

This is the 'bare-bones' structure for a Shiny app. As you get more complex, you may include other things in this folder, such as a data file, or the 'global.R' file, but that's further down the road.

Figure 3.1 File Structure for R Shiny app

3.4 The user interface can be broadly divided into three categories:

• **Title Panel:** The content in the title panel is displayed as metadata, as in top left corner of above image which generally provides name of the application and some other relevant information.

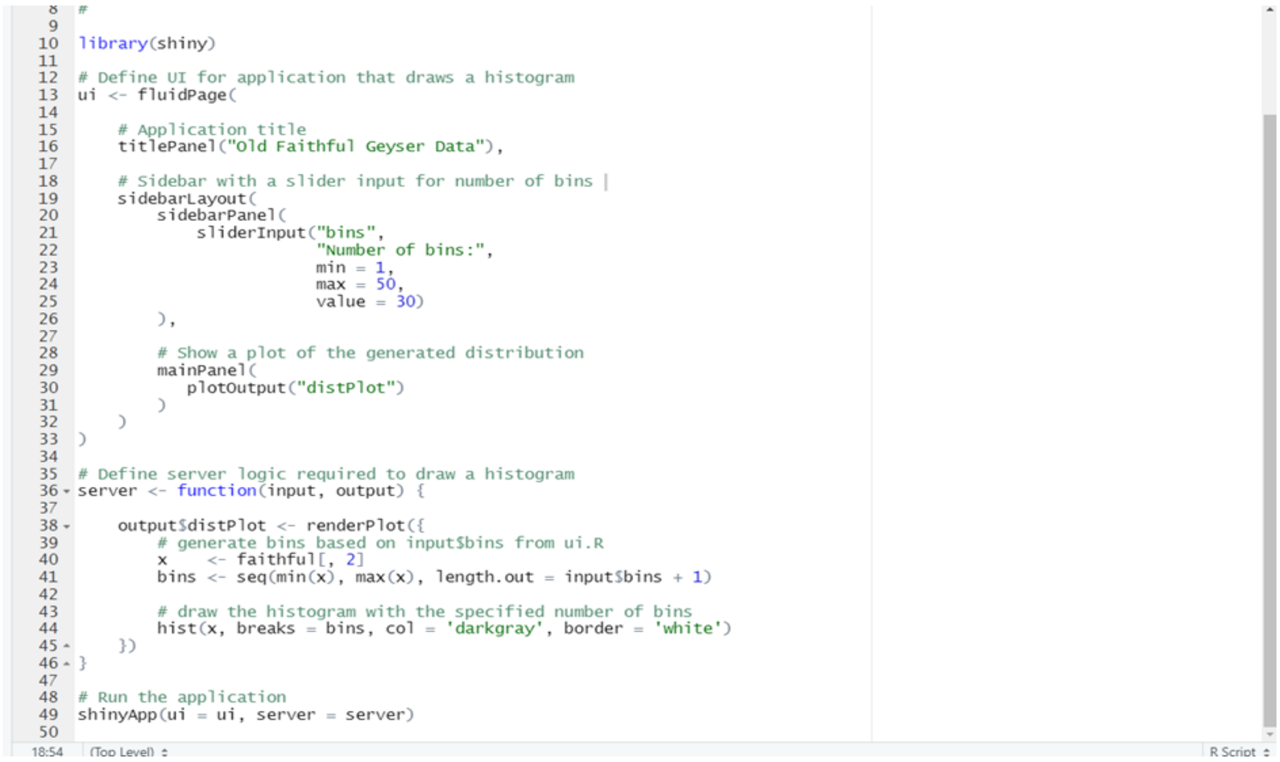
• **Sidebar Layout:** Sidebar layout takes input from the user in various forms like text input, checkbox input, radio button input, drop down input, etc. It is represented in dark background in left section of the above image.

• **Main Panel:** It is part of screen where the output(s) generated as a result of performing a set of operations on input(s) at the server.R is / are displayed.

3.4.1 Let's show the basic example of shiny,

This is the default shiny app in r studio. For running this app simply go in

file → shiny web application → give application name → click create.



```r
#

library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Figure 3.2 R Code of Histogram

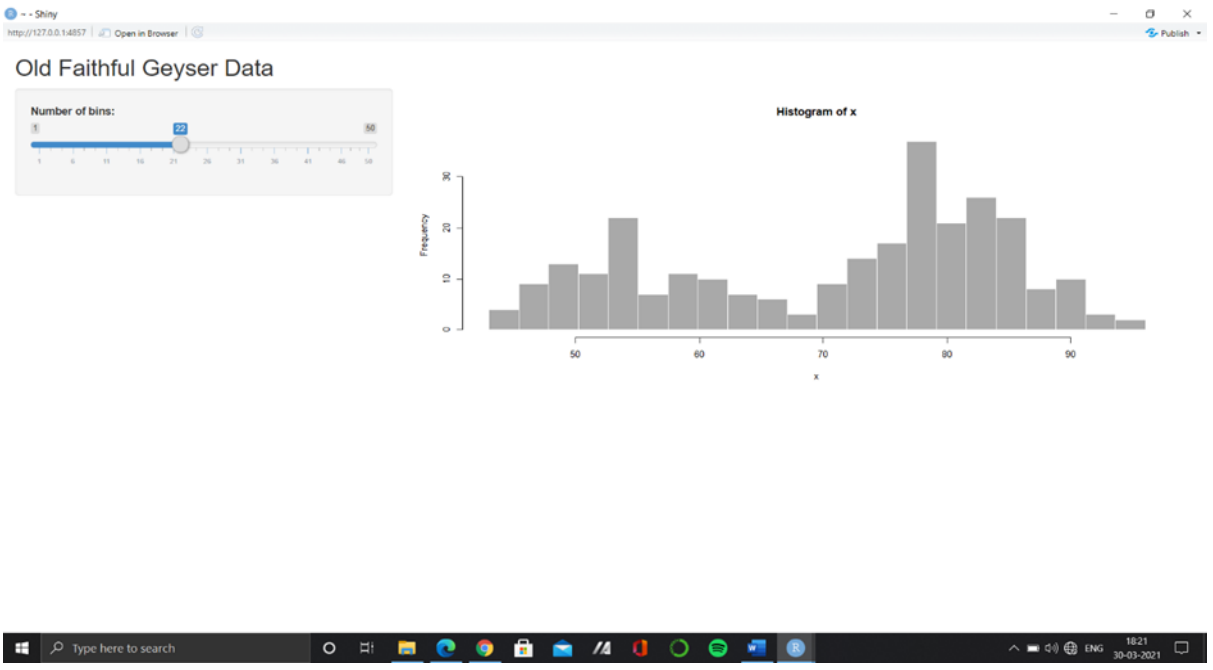
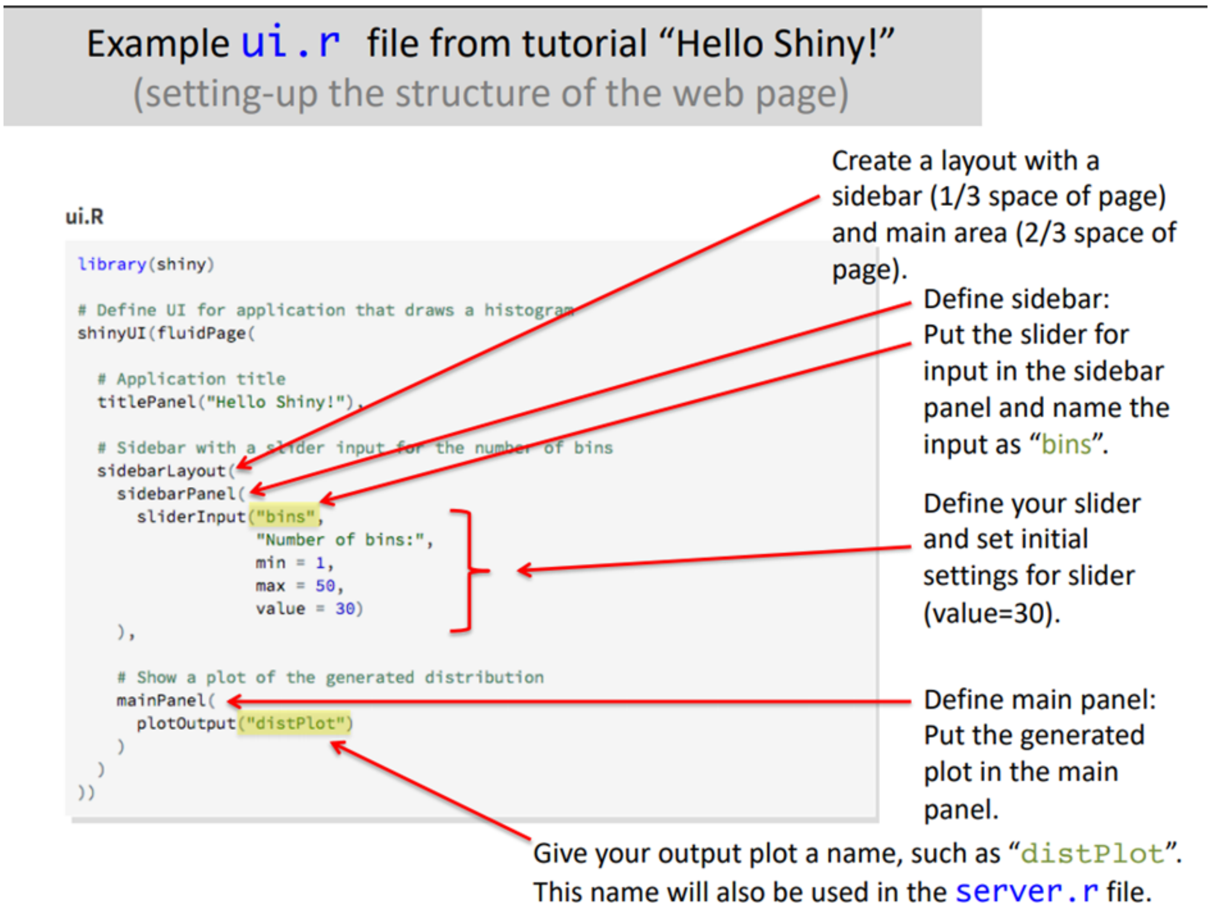


Figure 3.3 Output of Shiny App



Example ui.r file from tutorial “Hello Shiny!”
(setting-up the structure of the web page)

ui.R

```
library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

Create a layout with a sidebar (1/3 space of page) and main area (2/3 space of page).

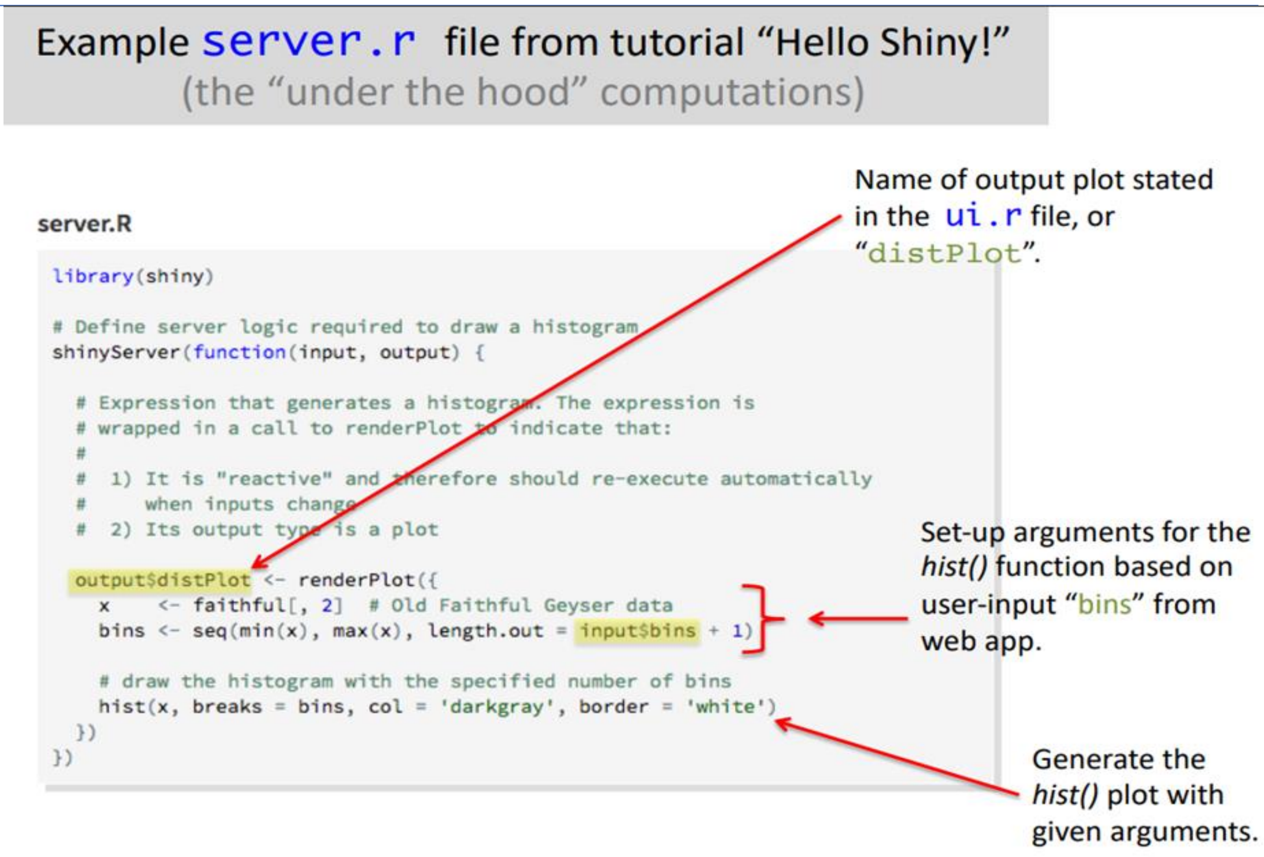
Define sidebar: Put the slider for input in the sidebar panel and name the input as “bins”.

Define your slider and set initial settings for slider (value=30).

Define main panel: Put the generated plot in the main panel.

Give your output plot a name, such as “distPlot”. This name will also be used in the server.r file.

Figure 3.4 Setting-up the structure of ui.R


Example server.r file from tutorial "Hello Shiny!"
(the "under the hood" computations)

server.R

```
library(shiny)

# Define server logic required to draw a histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  #  1) It is "reactive" and therefore should re-execute automatically
  #     when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
```

Name of output plot stated in the ui.r file, or "distPlot".
Set-up arguments for the hist() function based on user-input "bins" from web app.
Generate the hist() plot with given arguments.

Figure 3.5 Setting-up the structure of server.R

3.5 Live Reloading:

When you make changes to your underlying computer program definition or server script you don't must stop and restart your application to work out the changes. Simply save your changes and so reload the browser to work out the updated application in action. One qualification to this: when a browser reload occurs Shiny explicitly checks the timestamp of the app.R file to determine if it has to be re-sourced. Shiny isn't awake to other scripts or data files that change, so if you utilize those files and modify them, a point and restart of the applying is required. If you've got questions about this text or would really like to debate ideas presented here, please post on RStudio Community. Our developers monitor these forums and answer questions periodically. See help for more help with all things Shiny.

3.6 Inputs

Shiny apps include web elements called inputs that users can interact with by modifying their values. When the user changes the value of an input, other elements of the Shiny app that use the input value are updated. The Shiny apps in the previous examples contain several inputs. The first app shows a numeric input to introduce a number and a histogram that is built using this number. The second app shows an input to select a region and a bar plot that changes when the region is modified.

Shiny apps can include a variety of inputs that are useful for different purposes including texts, numbers and dates. These inputs can be modified by the user, and by doing this, the objects in the app that use them are updated.

3.6.1 Input Object Tips

- "choices" for inputs can be named using a list matching a display name to its value, such as list(Male = 'M', Female = 'F').
- Selective inputs are a useful option for long drop down lists.
- Always be aware of what the default value is for input objects you create

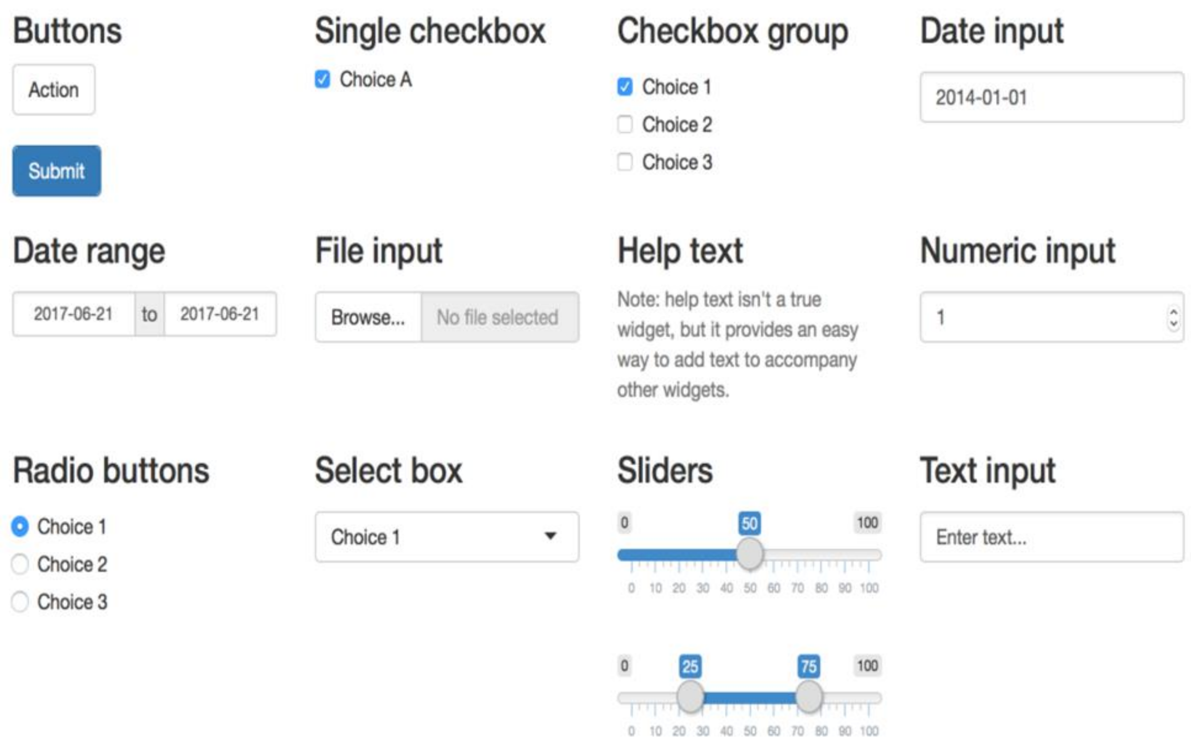


Figure 3.6 Input of Shiny App

To add an input to a Shiny app, we need to place an input function *Input() in the ui. Some examples of *Input() functions are

- textInput() which creates a field to enter text,
- dateRangeInput() which creates a pair of calendars for selecting a date range, and
- fileInput() which creates a control to upload a file.

An *Input() function has a parameter called inputId with the id of the input, a parameter called label with the text that appears in the app next to the input, and other parameters that vary from input to input depending on its purpose. For example, a numeric input where we can enter numbers can be created by writing numericInput(inputId = "n", label = "Enter a number", value = 25). This input has id n, label "Enter a number" and default value equal to 25. The value of a specific input can be accessed by writing input$ and the id of the input. For example, the value of the numeric input with id n can be accessed with input$n. We can build an output in the server() function using input$n. Each time the user enters a number in this input, the input$n value changes and the outputs that use it update.

3.7 Outputs

Shiny apps can include a variety of output elements including plots, tables, texts, images and HTML widgets. HTML widgets are objects for interactive web data visualizations created with JavaScript libraries. Examples of HTML widgets are interactive web maps created with the leaflet package and interactive tables created with the DT package. HTML widgets are embedded in Shiny by using the htmlwidgets package. We can use the values of inputs to construct output elements, and this causes the outputs to update each time input values are modified.

Shiny provides several output functions *Output() that turn R objects into outputs in the user interface. For example,

- textOutput() creates text,
- tableOutput() creates a data frame, matrix or other table-like structure, and imageOutput() creates an image.

The *Output() functions require an argument called outputId that denotes the id of the reactive element that is used when the output object is built in server(). A full list of input and output functions can be found in the Shiny reference guide.

Output objects are created by a render*() function in the server and displayed by a the paired *Output() function in the UI. The server function adds the result of each render*() function to a list of output objects.

Table No 3.1 Reactive Function Table

Desired UI Object	render*()	*Output()
Plot	renderPlot()	plotOutput()
Text	renderPrint()	verbatimTextOutput()
Text	renderText()	textOutput()
static table	renderTable()	tableOutput()
interactive table	renderDataTable()	dataTableOutput()

3.8 Dashboard

Installation: shinydashboard requires Shiny 0.11 or above. To install, run:

install.packages("shinydashboard")

Basics: A dashboard has three parts: a header, a sidebar, and a body. Here's the most minimal possible UI for a dashboard page.

```
## app.R ##
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
server <- function(input, output) { }
```

```
shinyApp(ui, server)
```

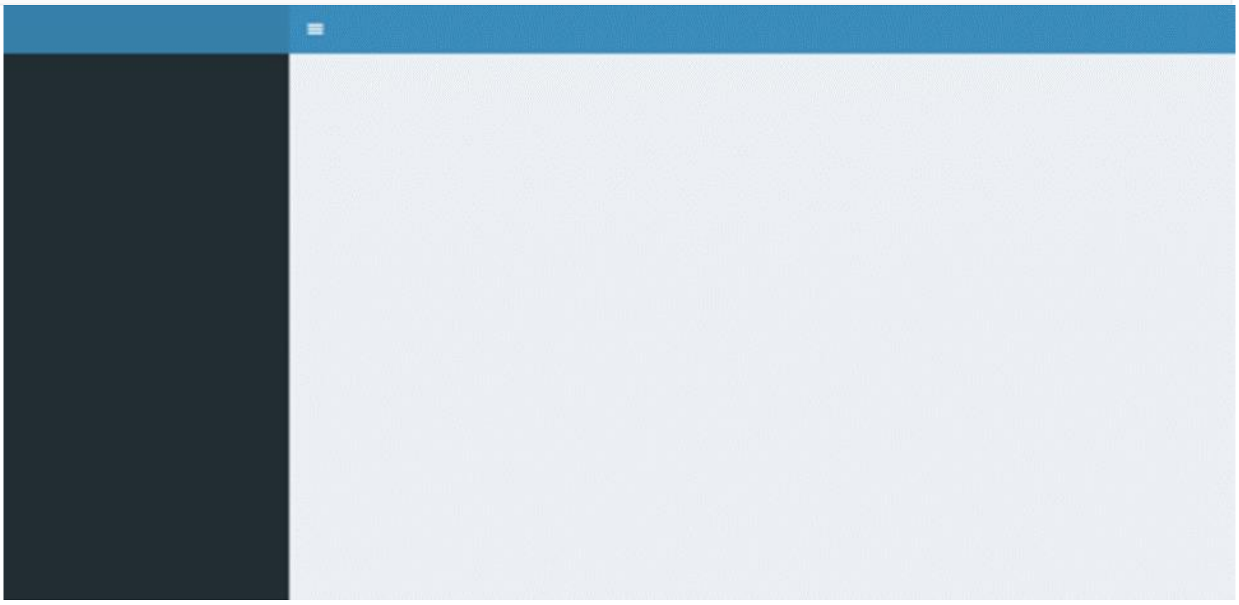


Figure 3.7 Blank Dashboard

Obviously, this dashboard isn't very useful. We'll need to add components that actually do something. In the body we can add boxes that have content.

Next, we can add content to the sidebar. For this example we'll add menu items that behave like tabs. These function similarly to Shiny's tabPanels: when you click on one menu item, it shows a different set of content in the main body.

There are two parts that need to be done. First, you need to add menuItems to the sidebar, with appropriate tabName

```
## Sidebar content
  dashboardSidebar(
      sidebarMenu(
      menuItem("Dashboard", tabName = "dashboard", icon = icon("dashboard")),
      menuItem("Widgets", tabName = "widgets", icon = icon("th"))
    ) )
```

In the body, add tabItems with corresponding values for tabName:

```
## Body content
  dashboardBody(
    tabItems(
      # First tab content
      tabItem(tabName = "dashboard",
        fluidRow(
          box(plotOutput("plot1", height = 250)),
          box(
            title = "Controls",
            sliderInput("slider", "Number of observations:", 1, 100, 50)
          )
        )
        ),
  # Second tab content
      tabItem(tabName = "widgets",
        h2("Widgets tab content")
      )))
```

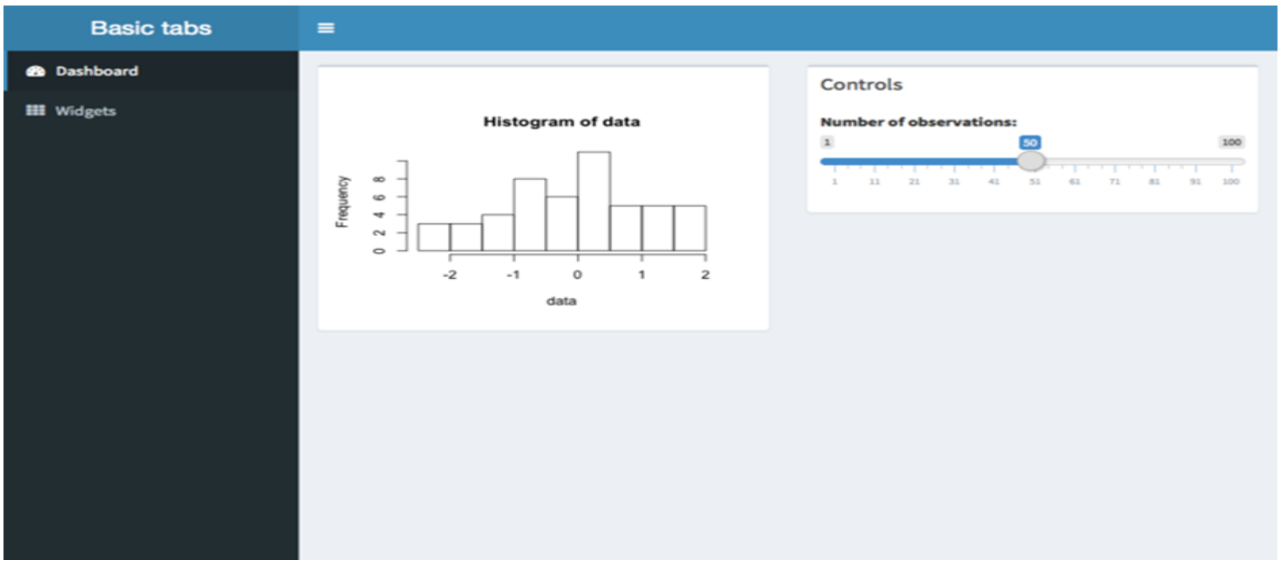


Figure 3.8 Dashboard

3.8.1 Structure overview

The dashboardPage() function expects three components: a header, sidebar, and body:

```
dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```

Now we'll look at each of the three main components of a shinydashboard.

3.8.1.1 Header

A header can have a title and dropdown menus. Here's an example:

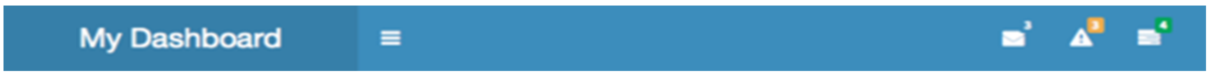


Figure 3.9 Dashboard Name

Setting the title is simple; just use the title argument. You can give your own name in header shows as the dashboradheader name.

```
dashboardHeader(title = "My Dashboard")
```

3.8.1.2 Sidebar

A sidebar is typically used for quick navigation. It can contain menu items that behave like tabs in a tabPanel, as well as Shiny inputs, like sliders and text inputs.

- **Sidebar menu items and tabs**

Links in the sidebar can be used like tabPanels from Shiny. That is, when you click on a link, it will display different content in the body of the dashboard. When the user clicks on one of the menu items, it switches content in the main body: menuItems have an option icon, which is created by the icon() function from Shiny. They also have an optional badge, with badgeLabel and badgeColor.

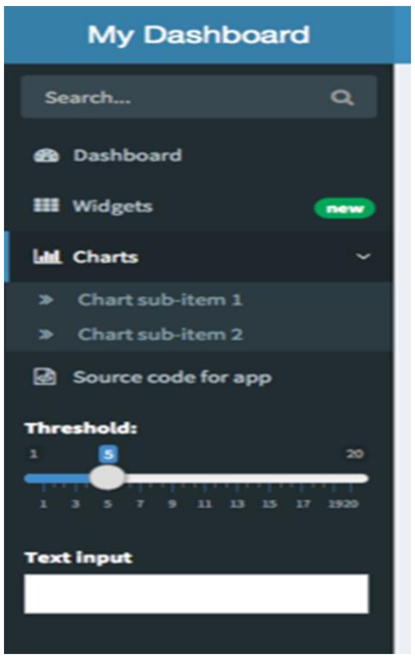


Figure 3.10 Sidebar

- **Inputs in the sidebar**

  A sidebar can also contain ordinary inputs, like sliderInputs and textInputs. shinydashboard also includes a special type of input, a sidebarSearchForm, which is on top in the screenshot above. This is essentially a specially formatted text input and and actionButton which appears as a magnifying glass icon (the icon can be changed with the icon argument).

3.8.1.3 Body

The body of a dashboard page can contain any regular Shiny content. However, if you're creating a dashboard you'll likely want to make something that's more structured. The basic building block of most dashboards is a box. Boxes in turn can contain any content.

3.8.1.3.1Boxes

Boxes are the main building blocks of dashboard pages. A basic box can be created with the box() function, and the contents of the box can be (most) any Shiny UI content. Box header color and title You can have solid headers with solidHeader=TRUE, and display a button in the upper right that will collapse the box with collapsible=TRUE.

```
box(
  title = "Histogram", status = "primary", solidHeader = TRUE,
  collapsible = TRUE,
  plotOutput("plot3", height = 250)
),
box(
  title = "Inputs", status = "warning", solidHeader = TRUE,
  "Box content here", br(), "More box content",
  sliderInput("slider", "Slider input:", 1, 100, 50),
  textInput("text", "Text input:")
```

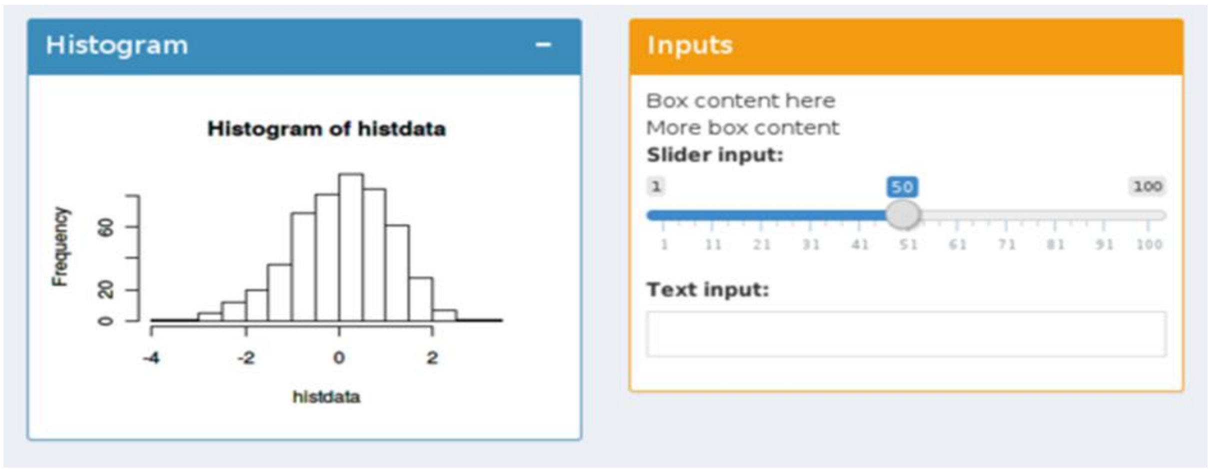


Figure 3.11 Box

3.8.1.3.2 tabBox

If you want a box to have tabs for displaying different sets of content, you can use a tabBox. A tabBox is similar to a tabsetPanel from Shiny in that it takes tabPanels as inputs, allows you to choose which tab is selected, and can be assigned an id. If id is present, you can access which tab is selected from the server. A tabBox also has similarities to a regular box from shinydashboard, in that you can control the height, width, and title. You can also choose which side the tabs appear on, with the side argument. Note that if side=”right”, the tabs will be displayed in reverse order.

3.8.2 Skins

There are a number of colour themes, or skins. The default is blue, but there are also black, purple, green, red, and yellow. You can choose which theme to use with dashboardPage(skin = "blue"), dashboardPage(skin = "black"), and so on.

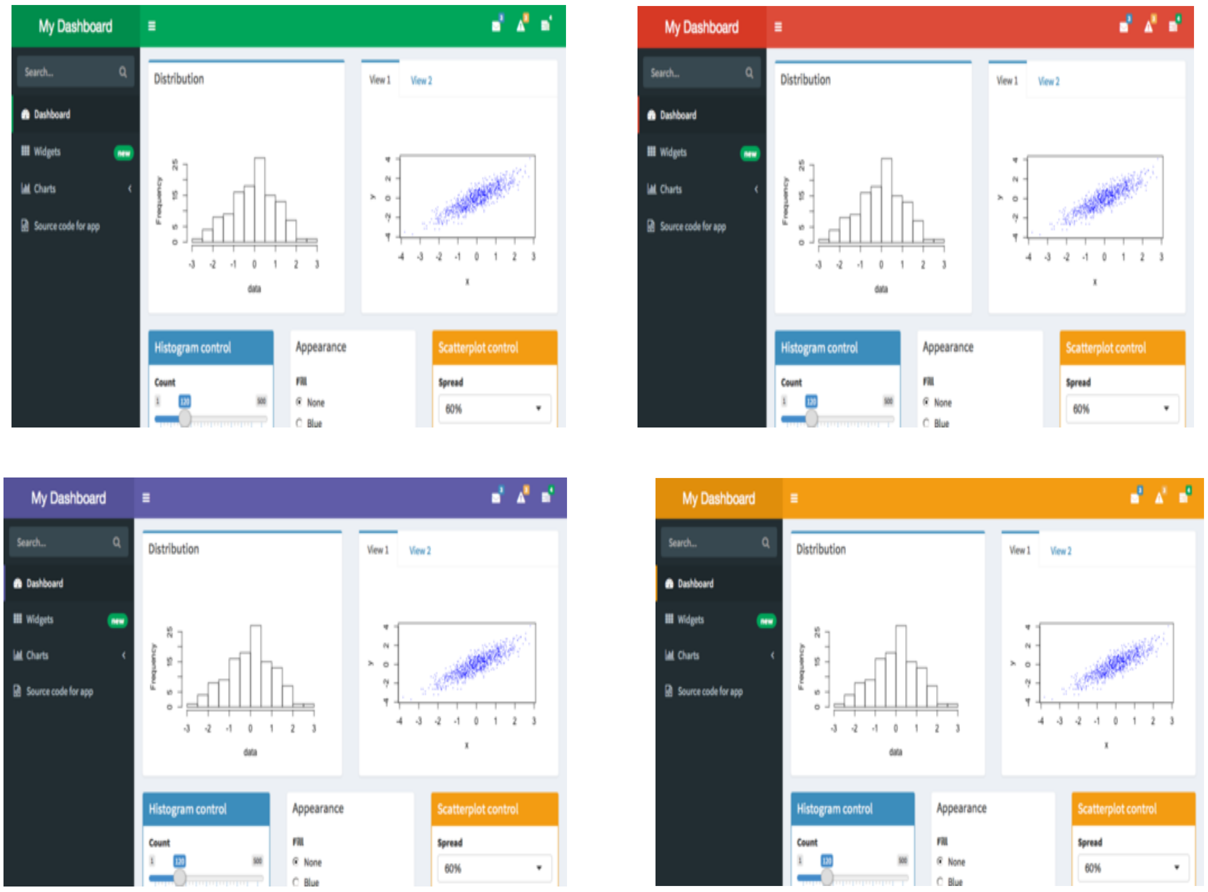


Figure 3.12 Skins of dashboard

4. THEORY

4.1 What Are Descriptive Statistics?

Descriptive statistics are brief descriptive coefficients that summarize a given data set, which can be either a representation of the total or a sample of a population. Descriptive statistics are de-escalated into measures of central tendency and measures of variability (spread). Measures of central tendency include the mean, median, and mode, while measures of variability include variance, variance, minimum and maximum variables, kurtosis, and skewness.

Descriptive statistics, in short, help describe and understand the features of a specific data set by giving short summaries about the sample and measures of the knowledge. the foremost recognized styles of descriptive statistics are measures of center: the mean, median, and mode, which are used at the foremost levels of math and statistics.

- Descriptive statistics summarizes or describes the characteristics of a data set.
- Descriptive statistics consists of two basic categories of measures: measures of central tendency and measures of variability (or spread).
- Measures of central tendency describe the center of a data set.
- Measures of variability or spread describe the dispersion of data within the set.

4.2 Correlation

Correlation shows the strength of a relationship between two variables and is expressed numerically by the correlation coefficient. The correlation coefficient's values range between -1.0 and 1.0. A perfect positive correlation means that the correlation coefficient is x exactly 1. A perfect negative correlation means that two assets move in opposite directions, while a zero correlation implies no linear relationship at all.

The most familiar measure of dependence between two quantities is the Pearson product-moment correlation coefficient (PPMCC), or "Pearson's correlation coefficient", commonly called simply "the correlation coefficient".

The population correlation coefficient ϱx,y between two random variables x and y with expected values μx and μy and standard deviations σx and σy is defined as

For a sample:

$$r = \sum \frac{(xi-\bar{x})(yi-\bar{y})}{\sqrt{\sum(xi-\bar{x})^2 \ \sum(yi-\bar{y})^2}}$$

r = correlation coefficient

xi = values of the x-variable in a sample

x̅ = mean of the values of the x-variable

yi = values of the x-variable in a sample

y̅ = mean of the values of the y-variable

For Population:

$$\varrho x,y = \frac{cov(\,x,y)}{\sigma x \ \sigma y}$$

ϱ x,y = is the covariance

σx = standard deviation of x

σy = standard deviation of y

4.3 Scatter Plot:

A scatter plot will be used either when one continuous variable that's under the control of the experimenter and therefore the other depends on that or when both continuous variables are independent. If a parameter exists that's systematically incremented and/or decremented by the opposite, it's called the control parameter or variable quantity and is customarily plotted along the horizontal axis. The measured or variable quantity is customarily plotted along the vertical axis. If no variable exists, both kind of variable are often plotted on either axis and a scatter plot will illustrate only the degree of correlation (not causation) between two variables.

A scatter plot can suggest various styles of correlations between variables with a particular confidence interval. as an example, weight and height, weight would get on y axis and height would air the x axis. Correlations could also be positive (rising), negative (falling), or null (uncorrelated). If the pattern of dots slopes from lower left to upper right, it indicates a correlation between the variables being studied. If the pattern of dots slopes from upper left to lower right, it indicates a indirect correlation. A line of best fit (alternatively called 'trendline') is drawn so as to review the connection between the variables. An equation for the correlation between the variables are often determined by established best-fit procedures. For a linear correlation, the best-fit procedure is understood as statistical regression and is certain to generate an accurate solution during a finite time. No universal best-fit procedure is absolute to generate an accurate solution for arbitrary relationships. A scatter plot is additionally very useful when we wish to determine how two comparable data sets comply with show nonlinear relationships between variables. the flexibility to try and do this could be enhanced by adding a smooth line like LOESS. Furthermore, if the information are represented by a mix model of straightforward relationships, these relationships are going to be visually evident as superimposed patterns. The scatter diagram is one in every of the seven basic tools of internal control.

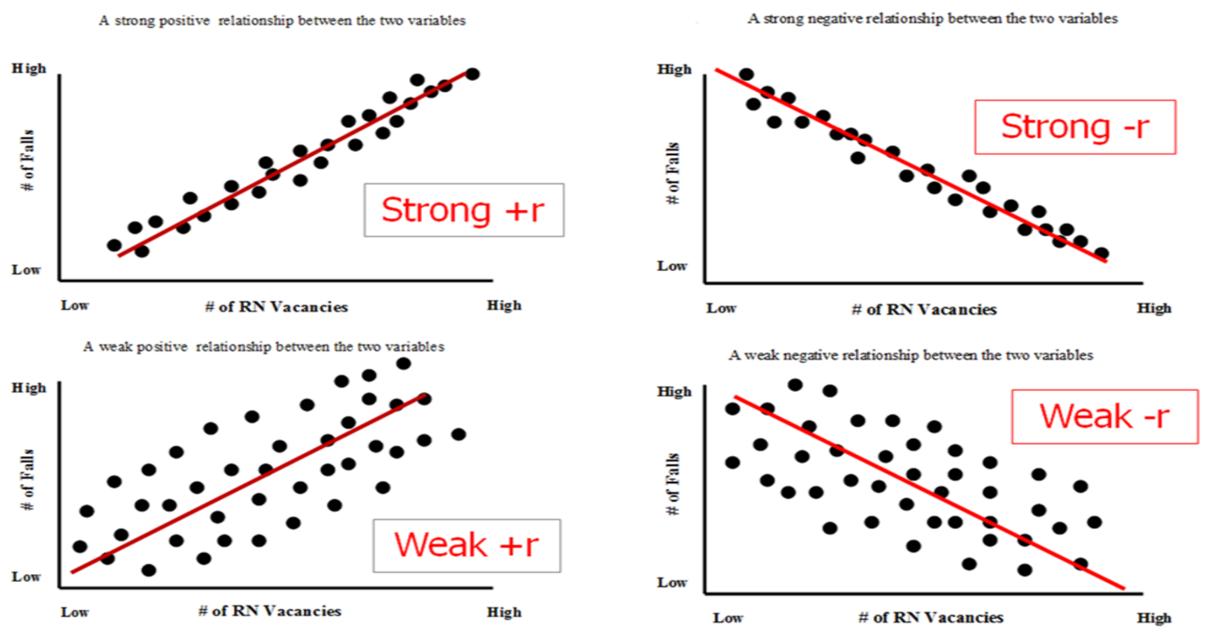


Figure 4.1 Type of Scatter plot

4.4 Box Plot:

Box plots (also called box-and-whisker plots or box-whisker plots) give a good graphical image of the concentration of the data. They also show how far the extreme values are from most of the data. A box plot is constructed from five values: the minimum value, the first quartile, the median, the third quartile, and the maximum value. We use these values to compare how close other data values are to them.

To construct a box plot, use a horizontal or vertical number line and a rectangular box. The smallest and largest data values label the endpoints of the axis. The first quartile marks one end of the box and the third quartile marks the other end of the box. Approximately the middle 50 percent of the data fall inside the box. The “whiskers” extend from the ends of the box to the smallest and largest data values. The median or second quartile can be between the first and third quartiles, or it can be one, or the other, or both. The box plot gives a good, quick picture of the data. Ex. The first quartile is two, the median is seven, and the third quartile is nine. The smallest value is one, and the largest value is 11.511.5. The following image shows the constructed box plot.

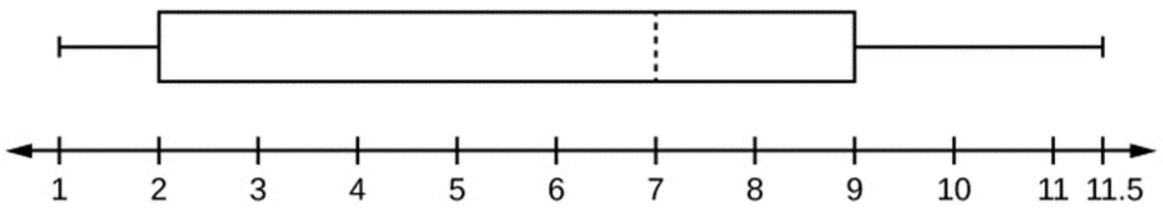


Figure 4.2 Boxplot

4.5 Histogram:

A frequency distribution shows how often each different value in a set of data occurs. A histogram is the most commonly used graph to show frequency distributions. It looks very much like a bar chart, but there are important differences between them. This helpful data collection and analysis tool is considered one of the seven basic quality tools.

Use a histogram when:

- The data are numerical
- You want to see the shape of the data's distribution, especially when determining whether the output of a process is distributed approximately normally.
- Analyzing whether a process can meet the customer's requirements
- Analyzing what the output from a supplier's process looks like
- Seeing whether a process change has occurred from one time period to another
- Determining whether the outputs of two or more processes are different
- You wish to communicate the distribution of data quickly and easily to other

Some possible descriptions of histograms

– Symmetric

– Skewed (asymmetric, long tail to one side) Right-tail stretched out... positive skew

Left-tail stretched out... negative skew

– Unimodal (one peak)

– Bimodal (two peaks)/

– Bell-shaped

– Uniformly distributed (flat)

Symmetric

If the distribution is symmetric,

the mean = median.

Right-skewed

If the distribution is right-skewed,

mean > median.

Left-skewed

If the distribution is left-skewed,

mean < median.

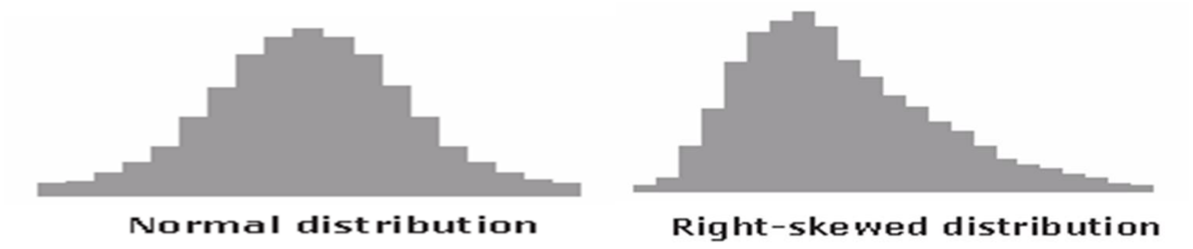


Figure 4.3 Type Histogram

4.6 Simple Linear Regression

Regression models describe the relationship between variables by fitting a line to the observed data. Linear regression models use a straight line, while logistic and nonlinear regression models use a curved line. Regression allows you to estimate how a dependent variable changes as the independent variable(s) change**.**

Simple linear regression is used to estimate the relationship between two quantitative variables. You can use simple linear regression when you want to know:

1. How strong the relationship is between two variables (e.g. the relationship between rainfall and soil erosion).

2. The value of the dependent variable at a certain value of the independent variable (e.g. the amount of soil erosion at a certain level of rainfall).

4.6.1 Assumptions of simple linear regression

1. Homogeneity of variance (homoscedasticity): the size of the error in our prediction doesn't change significantly across the values of the independent variable.

2. Independence of observations: the observations in the dataset were collected using statistically valid sampling methods, and there are no hidden relationships among observations.

3. Normality: The data follows a normal distribution.

4. The relationship between the independent and dependent variable is linear: the line of best fit through the data points is a straight line (rather than a curve or some sort of grouping factor).

If your data do not meet the assumptions of homoscedasticity or normality, you may be able to use a nonparametric test instead, such as the Spearman rank test.

How to perform a simple linear regression & Simple linear regression formula is

$$y = \beta_0 + \beta_1 X + \varepsilon$$

- y is the predicted value of the dependent variable (y) for any given value of the independent variable (x).
- B0 is the intercept, the predicted value of y when the x is 0.
- B1 is the regression coefficient – how much we expect y to change as x increases.
- x is the independent variable (the variable we expect is influencing y).
- e is the error of the estimate, or how much variation there is in our estimate of the regression coefficient.

Linear regression finds the line of best fit line through your data by searching for the regression coefficient (B1) that minimizes the total error (e) of the model.

4.6.2 Simple linear regression in R

R code for simple linear regression

income.happiness.lm <- lm(happiness ~ income, data = income.data).

This code takes the data you have collected **data = income.data** and calculates the effect that the independent variable **income** has on the dependent variable **happiness** using the equation for the linear model***:* lm().**

4.6.3 Interpreting the results

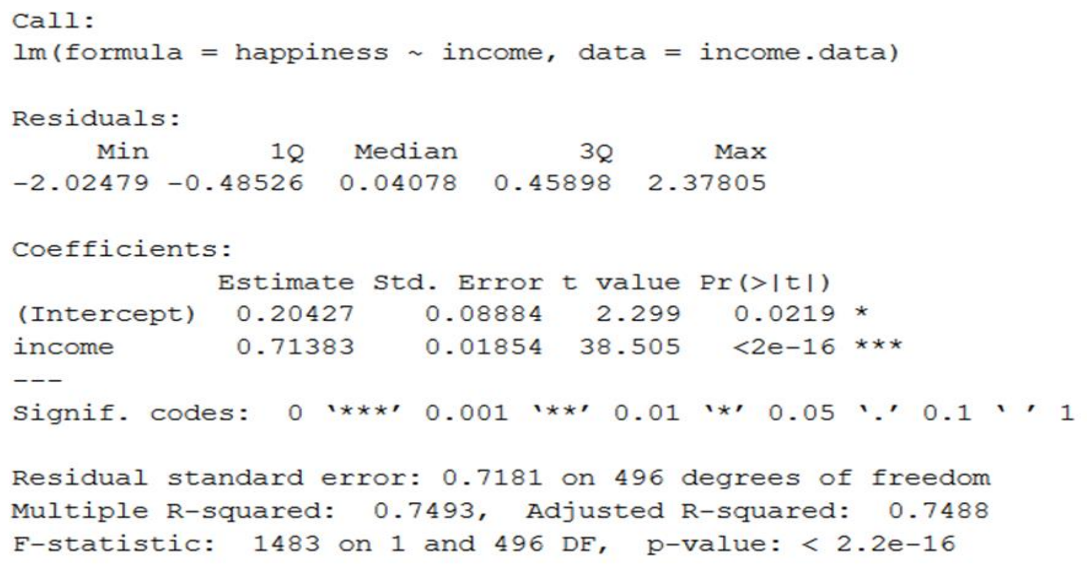
To view the results of the model, you can use the **summary ()** function in R:

summary(income.happiness.lm)

This function takes the most important parameters from the linear model and puts them into a table, which looks like this:

The next row in the 'Coefficients' table is income. This is the row that describes the estimated effect of income on reported happiness:

The Estimate column is the estimated effect, also called the regression coefficient or r2 value. The number in the table (0.713) tells us that for every one unit increase in income (where one unit of income = $10,000) there is a corresponding 0.71-unit increase in reported happiness (where happiness is a scale of 1 to 10).



```
Call:
lm(formula = happiness ~ income, data = income.data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.02479 -0.48526  0.04078  0.45898  2.37805

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.20427    0.08884   2.299   0.0219 *
income       0.71383    0.01854  38.505   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7181 on 496 degrees of freedom
Multiple R-squared:  0.7493,	Adjusted R-squared:  0.7488
F-statistic:  1483 on 1 and 496 DF,  p-value: < 2.2e-16
```

The **Std. Error** column displays the standard error of the estimate. This number shows how much variation there is in our estimate of the relationship between income and happiness.

The **t value** column displays the test statistic. Unless you specify otherwise, the test statistic used in linear regression is the t-value from a two-sided t-test. The larger the test statistic, the less likely it is that our results occurred by chance.

The **Pr(>| t |)** column shows the p-value. This number tells us how likely we are to see the estimated effect of income on happiness if the null hypothesis of no effect were true.

Because the p-value is so low ($p < 0.001$), we can **reject the null hypothesis** and conclude that income has a statistically significant effect on happiness.

The last three lines of the model summary are statistics about the model as a whole. The most important thing to notice here is the p-value of the model. Here it is significant ($p < 0.001$), which means that this model is a good fit for the observed data.

4.7 Multiple Linear Regression

4.7.1What is multiple Linear Regression (MLR)?

Multiple statistical regression (MLR), conjointly better-known merely as multivariate analysis, could be a applied math technique that uses many instructive variables to predict the result of a response variable. The goal of multiple statistical regression (MLR) is to model the linear relationship between the instructive (independent) variables and response (dependent) variable. In essence, multiple regression is the extension of ordinary least-squares (OLS) regression because it involves more than one explanatory variable.

KEY TAKEAWAYS

- Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable.
- Multiple regression is an extension of linear (OLS) regression that uses just one explanatory variable.
- MLR is used extensively in econometrics and financial inference.

4.7.2 Formula and Calculation of Multiple Linear Regression

$$yi=\beta0+\beta1xi1+\beta2xi2+...+\beta pxip+\epsilon$$

Where, for $i=n$ observation:

yi=dependent variable

xi=explanatory variables

$\beta0$=y-intercept (constant term)

βp=slope coefficients for each explanatory variable

ϵ=the model’s error term (also known as the residuals)

4.7.3 The Multiple Linear Regression Model is based on the following assumption:

- There is a linear relationship between the dependent variables and the independent variables
- The independent variables are not too highly correlated with each other
- yi observations are selected independently and randomly from the population
- Residuals should be normally distributed with a mean of 0 and variance σ

5. USED PACKAGES IN R SOFTWARE

5.1 Shinydashboard :

Dashboardsidebar :- Create a dashboard sidebar.

Description: A dashboard sidebar typically contains a sidebarMenu, although it may also contain a sidebarSearchForm, or other Shiny input

Usage:

```
   dashboardSidebar(..., disable = FALSE, width = NULL, collapsed =
FALSE)
```

Arguments:

- ….. : items to put in sidebar.
- disable: if 'True' the side will be disable.
- width: The width of sidebar. This must either be a number which specifies the width in pixels, or a string that specifies the width in CSS unit.
- collapsed: If 'True', the sidebar will be collapsed on app startup.

5.2 ShinydashboardPlus:

Description: shinydashboardPlus is based on the idea of shinydashboard, the latter not compatible with shinydashboard (you cannot use shinydashboard and ygdashboard at the same time). With shinydashboardPlus you can still work with the shinydashboard classic functions and enrich your dashboard with all additional functions of shinydashboardPlus.

Usage:

```
shinydashboardPlusGallery()
```

5.3 Shiny:

Easily build rich and productive interactive web apps in R — no HTML/CSS/JavaScript required.

Features: An intuitive and extensible reactive programming model which makes it easy to transform existing R code into a "live app" where outputs automatically react to new user input.

- Compared to event-based programming, reactivity allows Shiny to do the minimum amount of work when input(s) change, and allows humans to more easily reason about complex MVC logic.
- A prebuilt set of highly sophisticated, customizable, and easy-to-use widgets (e.g., plots, tables, sliders, dropdowns, date pickers, and more).
- An attractive default look based on Bootstrap which can also be easily customized with the possible package or avoided entirely with more direct R bindings to HTML/CSS/JavaScript.
- Seamless integration with R Markdown, making it easy to embed numerous applications natively within a larger dynamic document.
- Tools for improving and monitoring performance, including native support for async programming, caching, load testing, and more.
- Modules: a framework for reducing code duplication and complexity.
- An ability to bookmark application state and/or generate code to reproduce output(s).
- A rich ecosystem of extension packages for more custom widgets, input validation, unit testing, and more.

5.3.1 Functions in shiny:

a) ConditionalPanel:

Creates a panel that is visible or not, depending on the value of a JavaScript expression. The JS expression is evaluated once at startup and whenever Shiny detects a relevant change in input/output. In the JS expression, you can refer to ‘input’ and ‘output’ JavaScript objects that contain the current values of input and output. For example, if you have an input with an id of ‘foo’, then you can use ‘input.foo’ to read its value. (Be sure not to modify the input/output objects, as this may cause unpredictable behaviour.)

```
conditionalPanel(condition, ..., ns = NS(NULL))
```

Where,

ns: The ‘namespace’ object of the current module, is any.

b) DashboardHeader: Create a header for dashboard page. A dashboard header can be left blank, or it can include dropdown menu items on the right side.

```
dashboardHeader(..., title = NULL, titleWidth = NULL, disable =
FALSE,

  .list = NULL)
```

- Items to put in the header. Should be ‘dropdownMenus’ s.
- Title: An optional title to show in the header bar.. By default, this will also be used as the title shown in the browser's title bar. If you want that to be different from the text in the dashboard header bar, set the ‘title’ in ‘dashboardPage’.
- Titlewidth: The width of the title area. This must either be a number which specifies the width in pixels, or a string that specifies the width in CSS units.
- Disable: If TRUE, don't display the header bar.
- . list: An optional list containing items to put in the header. Same as the ... arguments, but in list format. This can be useful when working with programmatically generated items.

***c)* RadioButtons:** A set of radio buttons that can be added to a UI definition. A set of radio buttons that can be added to a UI definition.

```
radioButtons(inputId,label,choices = NULL,selected = NULL,inline =
FALSE,width = NULL,choiceNames = NULL,choiceValues = NULL)
```

- inputId: The 'input' slot that will be used to access the value .
- label : Display label for the control, or 'Null' for no label
- choices: List of value to select from(if elements of the list named then that name rather than the value is displayed to the user).If the argument is provided, then 'choiceName' and 'choicevalue' must not be provided, and vice-versa.The value should string; other type (such as logical and numbers) will be coerced tstring.
- Selected: The initially selected value. If not specified, then it defaults to the first item in choices. To start with no items selected, use character(0).
- Inline: If TRUE, render the choices inline (i.e. horizontally)
- Width: The width of the input, e.g. '400px', or '100%';
- choiceNames, choic: List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length).
- dashboardSidebar: Create a dashboard sidebar. A dashboard sidebar typically contains a 'sidebarMenu', although it may also contain a 'sidebarSearchForm', or other Shiny inputs.

```
dashboardSidebar(..., disable = FALSE, width = NULL, collapsed = FALSE)
```

- … : Items to put in sidebar.
- disable: If 'True' the side will be disable.
- width: The width of sidebar. This must either be a number which specifies the width in pixels, or a string that specifies the width in CSS unit.
- collapsed: If 'True', the sidebar will be collapsed on app startup.

d) SidebarMenu: Create a dashboard sidebar menu and menu items. A dashboardSidebar can contain a sidebarMenu. A sidebarMenu contains menuItems, and they can in turn contain menuSubItems.

```
sidebarMenu(..., id = NULL, .list = NULL)

menuItem(text, ..., icon = NULL, badgeLabel = NULL, badgeColor = "green",

 tabName = NULL, href = NULL, newtab = TRUE, selected = NULL,

 expandedName = as.character(gsub("[[:space:]]", "", text)),

 startExpanded = FALSE)

menuSubItem(text, tabName = NULL, href = NULL, newtab = TRUE,

 icon = shiny::icon("angle-double-right"), selected = NULL)
```

- .. : For menu items, this may consist of menuSubItems.
- Id: For sidebarMenu, if id is present, this id will be used for a Shiny input value, and it will report which tab is selected. For example, if id="tabs", then input$tabs will be the tabName of the currently-selected tab. If you want to be able to bookmark and restore the selected tab, an id is required.
- list: An optional list containing items to put in the menu Same as the ... arguments, but in list format. This can be useful when working with programmatically generated items.
- Text: Text to show for the menu item.
- Icon: An icon tag, created by icon. If NULL, don't display an icon.
- badgeLabel: A label for an optional badge. Usually a number or a short word like "new".
- badgeColor: A color for the badge. Valid colors are listed in validColors.
- tabName: The name of a tab that this menu item will activate. Not compatible with href.
- Newtab: If href is supplied, should the link open in a new browser tab?
- Selected: If TRUE, this menuItem or menuSubItem will start selected. If no item have selected=TRUE, then the first menuItem will start selected.
- expandedName: A unique name given to each menuItem that serves to indicate which one (if any) is currently expanded. (This is only applicable to menuItems that have children and it is mostly only useful for bookmarking state.)
- startExpanded: Should this menuItem be expanded on app startup? (This is only applicable to menuItems that have children, and only one of these can be expanded at any given time).

e) FileInput: File inputs. Upload files to the server.

```
fileInput( id, placeholder = "Choose file", browse = "Browse", ..., multiple = TRUE, accept =
NULL)
```

- id: A character string specifying the id of the reactive input.
- Placeholder: A character string specifying the text inside the file input, defaults to "Choose file".
- Browse: A character string specifying the label of file input, defaults to "Browse".
- ...: Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
- Multiple: One of TRUE or FALSE specifying whether or not the user can upload multiple files at once, defaults to TRUE.
- Accept: A character vector of possible MIME types or file extensions, defaults to NULL, in which case any file type may be selected.

f) DashboardBody: The main body of a dashboard page. The main body typically contains boxes. Another common use pattern is for the main body to contain tabItems.

```
dashboardBody(...)
```

5.4 readxl:

The readxl package makes it easy to get data out of Excel and into R. Compared to many of the existing packages (e.g. gdata, xlsx, xlsReadWrite) readxl has no external dependencies so it's easy to install and use on all operating systems. It is designed to work with tabular data stored in a single sheet.

Readxl supports both the legacy .xls format and the modern xml-based .xlsx format. .xls support is made possible the with libxls C library, which abstracts away many of the complexities of the underlying binary format. To parse .xlsx, we use the RapidXML C++ library.

```
install.packages("readxl")
```

5.5 DT:

The R package DT provides an R interface to the JavaScript library DataTables. R data objects (matrices or data frames) can be displayed as tables on HTML pages, and DataTables provides filtering, pagination, sorting, and many other features in the tables. The main function in this package is datatable(). It creates an HTML widget to display R data objects with DataTables.

Currently, DT only supports the Bootstrap style besides the default style. You can use the argument style = 'bootstrap' to enable the Bootstrap style, and adjust the table classes accordingly using Bootstrap table class names, such as table-stripe and table-hover. Actually, DT will automatically adjust the class names even if you provided the DataTables class names such as stripe and hover. You can enable table editing using the argument editable (DT::datatable for its possible values). Then you will be able to double-click a cell to edit its value. It works in both client-side and server-side processing modes.

```
DT::datatable(head(iris), editable = 'cell')
```

5.6 dplyr:

Dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

mutate() adds new variables that are functions of existing variables

select() picks variables based on their names.

filter() picks cases based on their values.

summarise() reduces multiple values down to a single summary.

arrange() changes the ordering of the rows.

dplyr is designed to abstract over how the data is stored. That means as well as working with local data frames, you can also work with remote database tables, using exactly the same R code.

5.7 ggplot:

Create a new ggplot. ggplot() initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

```
ggplot(data = NULL, mapping = aes(), ..., environment = parent.frame())
```

- data: Default dataset to use for plot. If not already a data.frame, will be converted to one by fortify(). If not specified, must be supplied in each layer added to the plot.
- Mapping: Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
- ...: Other arguments passed on to methods. Not currently used.
- Environment: DEPRECATED. Used prior to tidy evaluation.

- ggplot() is used to construct the initial plot object, and is almost always followed by + to add component to the plot. There are three common ways to invoke ggplot:
- ggplot(df, aes(x, y, other aesthetics))
- ggplot(df)
- ggplot()

The first method is recommended if all layers use the same data and the same set of aesthetics, although this method can also be used to add a layer using data from another data frame. See the first example below. The second method specifies the default data frame to use for the plot, but no aesthetics are defined up front. This is useful when one data frame is used predominantly as layers are added, but the aesthetics may vary from one layer to another. The third method initializes a skeleton ggplot object which is fleshed out as layers are added. This method is useful when multiple data frames are used to produce different layers, as is often the case in complex graphics.

5.8 ggvis:

Description: ggvis is used to turn a dataset into a visualisation, setting up default mappings between variables in the dataset and visual properties. Nothing will be displayed until you add additional layers.

Usage:

```
ggvis(data = NULL, ..., env = parent.frame())
```

Arguments:

- Data: A data object.
- ... :   Property mappings. If not named, the first two mappings are taken to be x and y.Common properties are x, y, stroke, fill, opacity, shape
- env: Environment in which to evaluate properties.

5.9 corrplot

Description: A graphical display of a correlation matrix, confidence interval. The details are paid great attention to. It can also visualize a general matrix by setting is.corr = FALSE.

Usage:

```
corrplot(

  corr,

  method = c("circle", "square", "ellipse", "number", "shade", "color", "pie"),

  type = c("full", "lower", "upper"),
  add = FALSE,

  col = NULL,

  full_col = TRUE )
```

5.10 ggally

Extension to ggplot2. ggplot2 is a plotting system for R based on the grammar of graphics. GGally extends ggplot2 by adding several functions to reduce the complexity of combining geoms with transformed data. Some of these functions include a pairwise plot matrix, a scatterplot plot matrix, a parallel coordinates plot, a survival plot, and several functions to plot networks.

5.11 stats

Description: Various summary statistics are calculated for different types of data**.**

```
stats(x, by)
```

Arguments:

- x: The data structure to compute the statistics. This can either be a vector, matrix (data sets are the columns), or a list (data sets are the components).
- By: If x is a vector, an optional vector (either character or numerical) specifying the categories to divide x into separate data sets.

5.12 ggpubr

Description: An excellent and flexible package for elegant data visualization in R. However, the default generated plots require some formatting before we can send them for publication. Furthermore, to customize a ggplot, the syntax is opaque and this raises the level of difficulty for researchers with no advanced R programming skills.

The 'ggpubr' package provides some easy-to-use functions for creating and customizing 'ggplot2'- based publication ready plots.

5.13 shinyWidgets

Description: This package offers custom widgets and other components to enhance your shiny applications. You can replace classical checkboxes with switch button, add colors to radio buttons and checkbox group, use buttons as radio or checkboxes. Each widget has an update method to change the value of an input from the server.

Usage:

```
shinyWidgetsGallery()
```

5.14 Plotly

Description: Create interactive web graphics from ‘ggplot2’ graphs and a custom interface to the JavaScript library ‘plotly.js’ inspired by the grammar of graphics. This function is now deprecated. It used to provide a way to store plotly account credentials, but that can now be done with environment variables.

Usage:

```
Plotly(username, key)
```

6. FLOW CHART OF APPLICATION

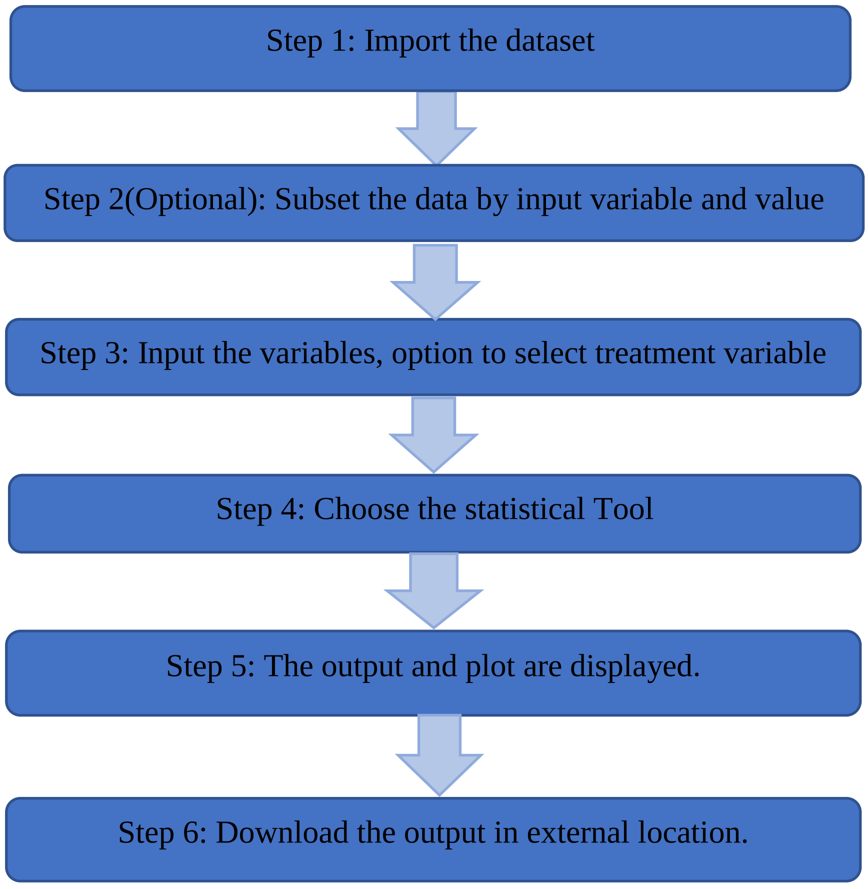

Step 1: Import the dataset
Step 2(Optional): Subset the data by input variable and value
Step 3: Input the variables, option to select treatment variable
Step 4: Choose the statistical Tool
Step 5: The output and plot are displayed.
Step 6: Download the output in external location.

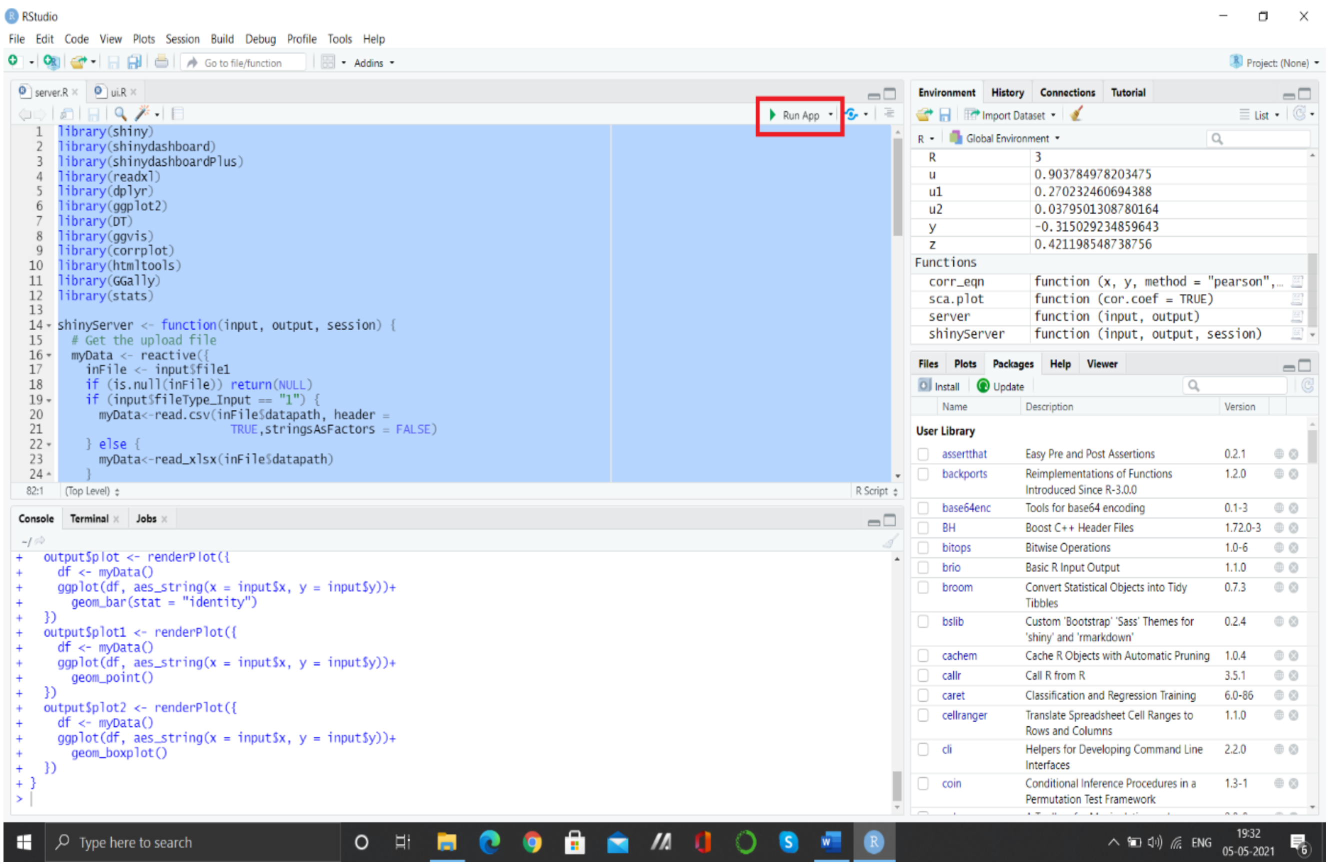
Figure 6.1 Flow chart of Application

7. VISUALIZATION

7.1 HOW TO USE APPLICATION

From above theory about shiny application, we seen how to build shiny ui (user interface) file and server file for application. Now we have created shiny application for statistical analysis, we seen how to use it by step by step.

Step 1: Run ui and server file and click on **Run app** button in R.



```r
library(shiny)
library(shinydashboard)
library(shinydashboardPlus)
library(readxl)
library(dplyr)
library(ggplot2)
library(DT)
library(ggvis)
library(corrplot)
library(htmltools)
library(GGally)
library(stats)

shinyServer <- function(input, output, session) {
  # Get the upload file
  myData <- reactive({
    inFile <- input$file1
    if (is.null(inFile)) return(NULL)
    if (input$fileType_Input == "1") {
      myData<-read.csv(inFile$datapath, header =
                         TRUE,stringsAsFactors = FALSE)
    } else {
      myData<-read_xlsx(inFile$datapath)
    }
```

```
+   output$plot <- renderPlot({
+     df <- myData()
+     ggplot(df, aes_string(x = input$x, y = input$y))+
+       geom_bar(stat = "identity")
+   })
+   output$plot1 <- renderPlot({
+     df <- myData()
+     ggplot(df, aes_string(x = input$x, y = input$y))+
+       geom_point()
+   })
+   output$plot2 <- renderPlot({
+     df <- myData()
+     ggplot(df, aes_string(x = input$x, y = input$y))+
+       geom_boxplot()
+   })
+ }
> 
```

R	3
u	0.903784978203475
u1	0.270232460694388
u2	0.0379501308780164
y	-0.315029234859643
z	0.421198548738756
Functions	
corr_eqn	function (x, y, method = "pearson",…
sca.plot	function (cor.coef = TRUE)
server	function (input, output)
shinyServer	function (input, output, session)

Name	Description	Version
User Library		
assertthat	Easy Pre and Post Assertions	0.2.1
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.2.0
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.72.0-3
bitops	Bitwise Operations	1.0-6
brio	Basic R Input Output	1.1.0
broom	Convert Statistical Objects into Tidy Tibbles	0.7.3
bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.2.4
cachem	Cache R Objects with Automatic Pruning	1.0.4
callr	Call R from R	3.5.1
caret	Classification and Regression Training	6.0-86
cellranger	Translate Spreadsheet Cell Ranges to Rows and Columns	1.1.0
cli	Helpers for Developing Command Line Interfaces	2.2.0
coin	Conditional Inference Procedures in a Permutation Test Framework	1.3-1

Figure 7.1 Window of R code for shiny app

The application open in browser and we see the web page like following picture shows the interface of application. Where the name of application is given is "**Data Visualization**" in header panel. After that we see sidebar, in that we see choose file option for upload your data in application for further analysis. We upload .csv as well as .xlsx file in application. If we have

.csv then select .csv option or if we have .xlsx then select .xlsx file. We see in sidebar panel raw data, data summary, histogram, plots, correlation, simple linear regression heading.

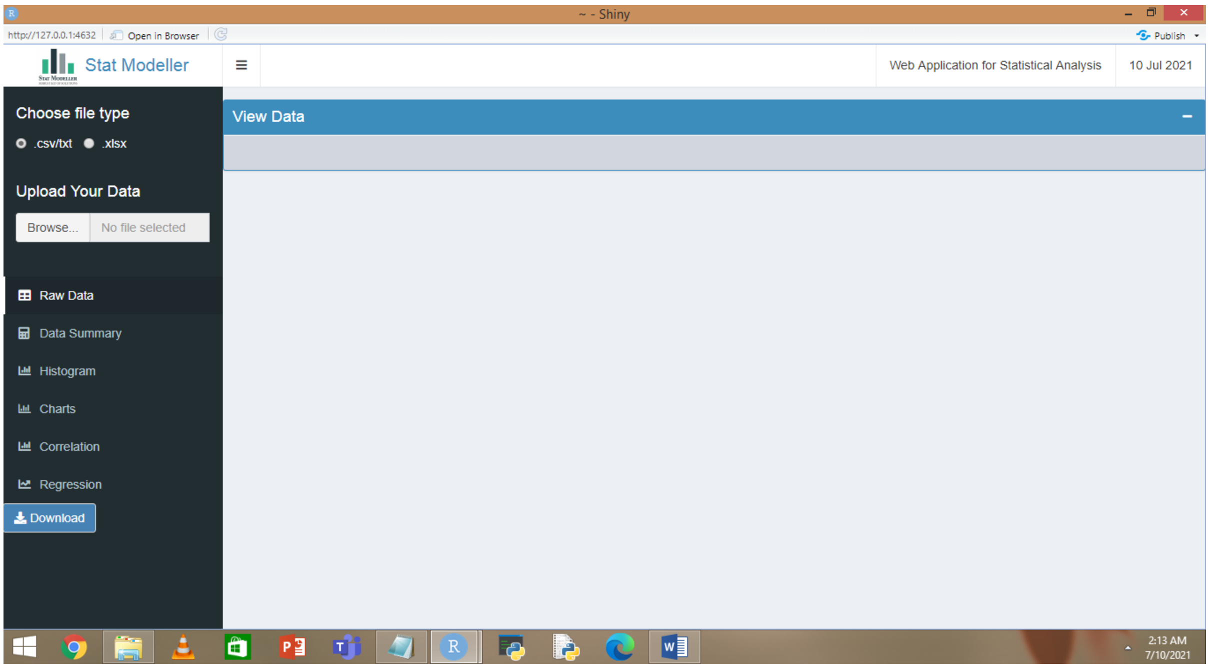
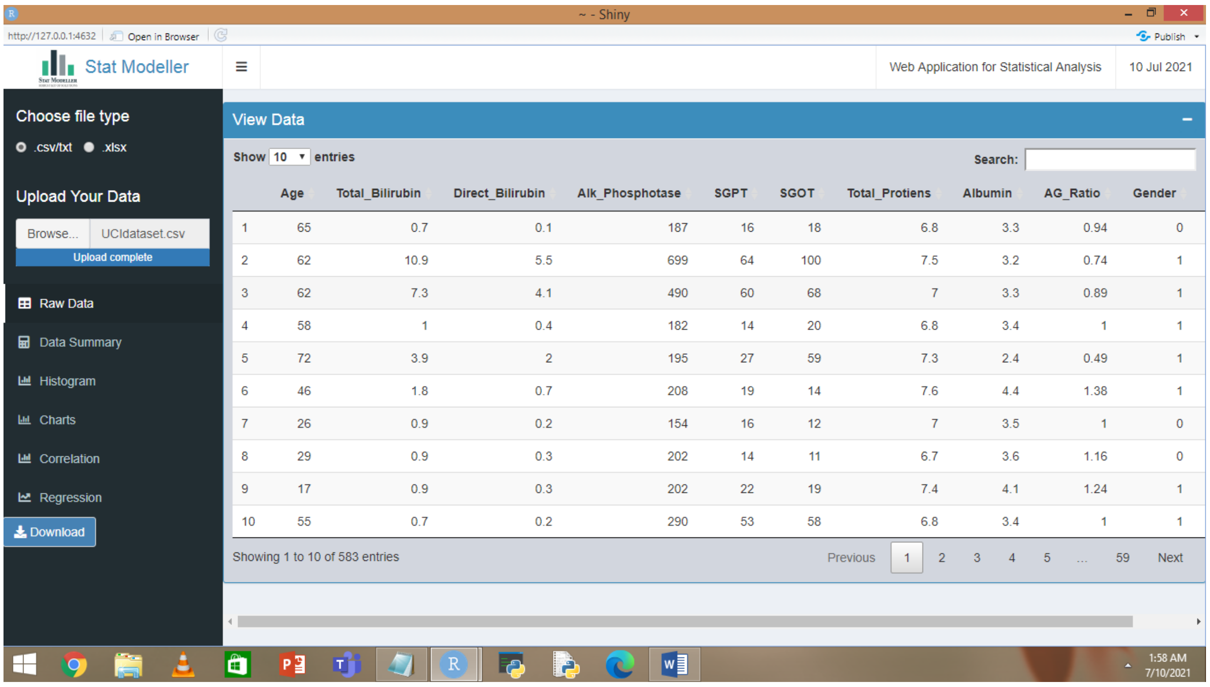


Figure 7.2 Window of shiny application

Step 2: After the run application we see the dashboard, first we upload our data file in application. If we have .csv select csv, if excel file select .xlsx.

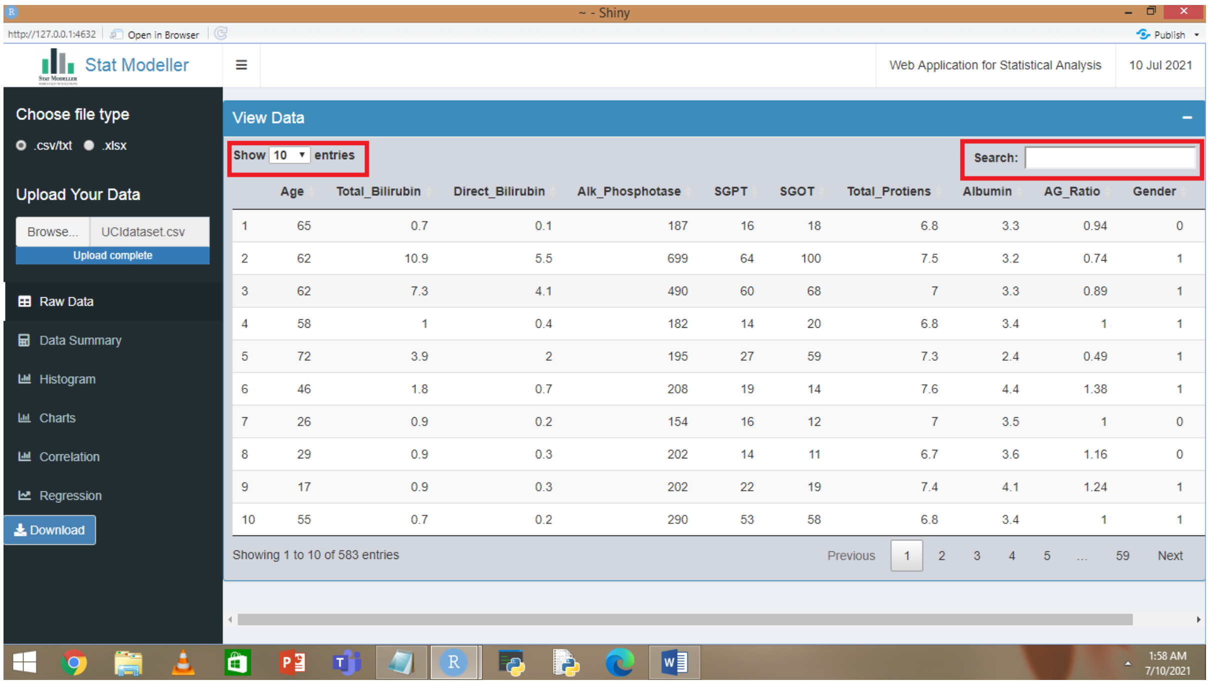


	Age	Total_Bilirubin	Direct_Bilirubin	Alk_Phosphotase	SGPT	SGOT	Total_Protiens	Albumin	AG_Ratio	Gender
1	65	0.7	0.1	187	16	18	6.8	3.3	0.94	0
2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1
3	62	7.3	4.1	490	60	68	7	3.3	0.89	1
4	58	1	0.4	182	14	20	6.8	3.4	1	1
5	72	3.9	2	195	27	59	7.3	2.4	0.49	1
6	46	1.8	0.7	208	19	14	7.6	4.4	1.38	1
7	26	0.9	0.2	154	16	12	7	3.5	1	0
8	29	0.9	0.3	202	14	11	6.7	3.6	1.16	0
9	17	0.9	0.3	202	22	19	7.4	4.1	1.24	1
10	55	0.7	0.2	290	53	58	6.8	3.4	1	1

Showing 1 to 10 of 583 entries   Previous 1 2 3 4 5 … 59 Next

Figure 7.3 Window of data upload in shiny application

You see that we select csv file and uploaded file is .csv to. After uploading data is upload in dashboard body. This is our output of application in that we see only first 10 data entries and all variable of data. If we want show first 50 observation then in upper panel there is show entries option select feasible observation. Search bar also shown for searching variable or observation.



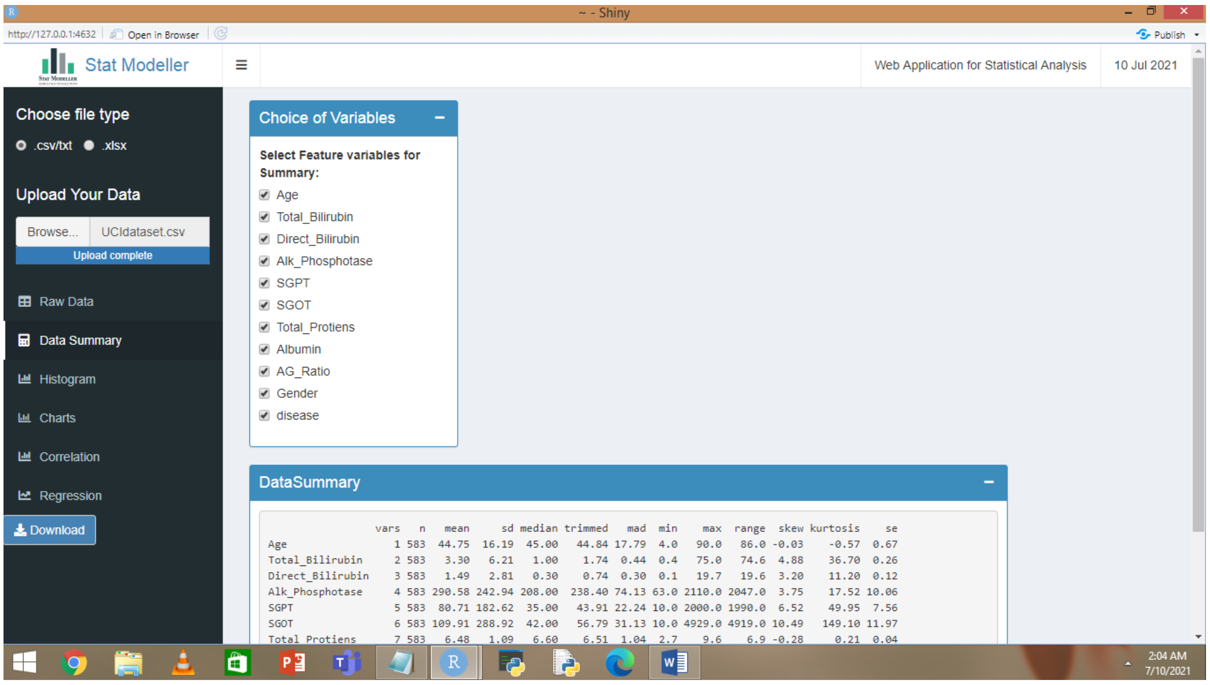
	Age	Total_Bilirubin	Direct_Bilirubin	Alk_Phosphotase	SGPT	SGOT	Total_Protiens	Albumin	AG_Ratio	Gender
1	65	0.7	0.1	187	16	18	6.8	3.3	0.94	0
2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1
3	62	7.3	4.1	490	60	68	7	3.3	0.89	1
4	58	1	0.4	182	14	20	6.8	3.4	1	1
5	72	3.9	2	195	27	59	7.3	2.4	0.49	1
6	46	1.8	0.7	208	19	14	7.6	4.4	1.38	1
7	26	0.9	0.2	154	16	12	7	3.5	1	0
8	29	0.9	0.3	202	14	11	6.7	3.6	1.16	0
9	17	0.9	0.3	202	22	19	7.4	4.1	1.24	1
10	55	0.7	0.2	290	53	58	6.8	3.4	1	1

Showing 1 to 10 of 583 entries

Previous 1 2 3 4 5 ... 59 Next

Figure 7.4 Window of search bar

Step 3: We seen the data summary of our upload data file. In data summary we have calculated mean, 1st quartile, median, 3rd quartile, minimum and maximum.



Select Feature variables for Summary:

- ☑ Age
- ☑ Total_Bilirubin
- ☑ Direct_Bilirubin
- ☑ Alk_Phosphotase
- ☑ SGPT
- ☑ SGOT
- ☑ Total_Protiens
- ☑ Albumin
- ☑ AG_Ratio
- ☑ Gender
- ☑ disease

```
                 vars   n   mean     sd median trimmed   mad  min    max  range  skew kurtosis    se
Age                 1 583  44.75  16.19  45.00   44.84 17.79  4.0   90.0   86.0 -0.03    -0.57  0.67
Total_Bilirubin     2 583   3.30   6.21   1.00    1.74  0.44  0.4   75.0   74.6  4.88    36.70  0.26
Direct_Bilirubin    3 583   1.49   2.81   0.30    0.74  0.30  0.1   19.7   19.6  3.20    11.20  0.12
Alk_Phosphotase     4 583 290.58 242.94 208.00  238.40 74.13 63.0 2110.0 2047.0  3.75    17.52 10.06
SGPT                5 583  80.71 182.62  35.00   43.91 22.24 10.0 2000.0 1990.0  6.52    49.95  7.56
SGOT                6 583 109.91 288.92  42.00   56.79 31.13 10.0 4929.0 4919.0 10.49   149.10 11.97
Total_Protiens      7 583   6.48   1.09   6.60    6.51  1.04  2.7    9.6    6.9 -0.28     0.21  0.04
```

Figure 7.5 Window of Data Summary

Step 4: Now we display charts for which variable are select in previous. We display 3 charts barplot, boxplot, scatter plot. This variable is used for barplot, boxplot, scatter plot. For selection of x-variable go on side click on there is all variable which is in data then select your variable for plot. Then goes y-variable select variable for y-axis for plot. Selection of variable is shown in form of plots in charts section which is shows in sidebar. And also shows the histogram of selected variable.

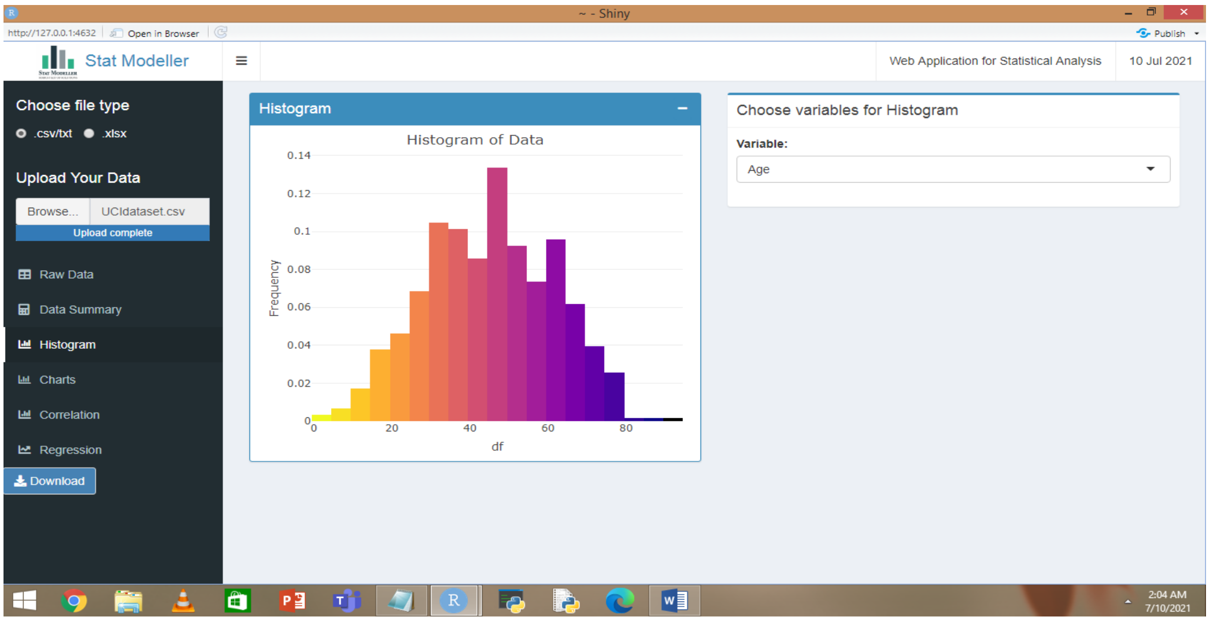


Figure 7.6 Window of Histogram for selected variable

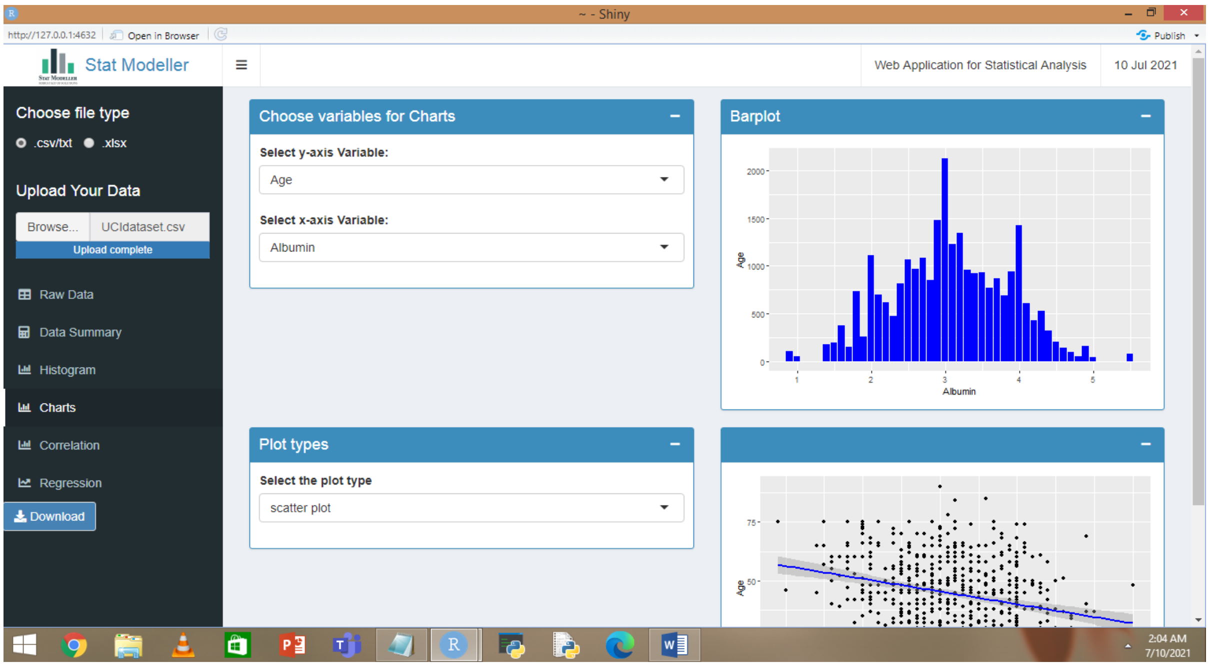
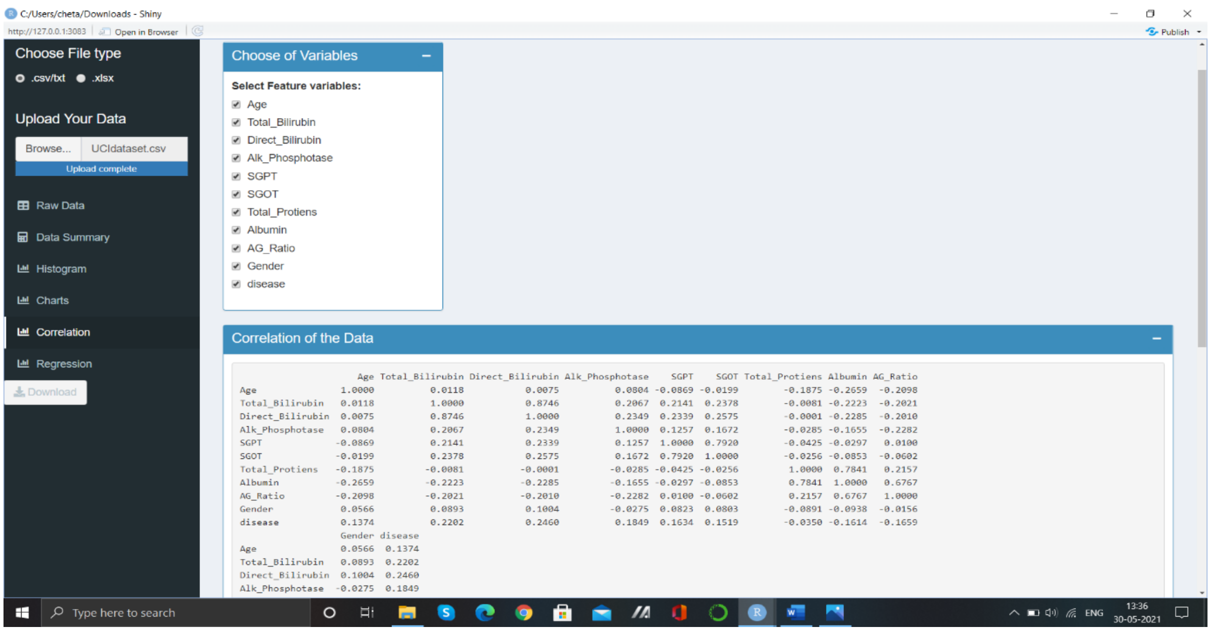


Figure 7.7 Window of charts for selected variable

Step 5: After charts we seen the correlation of data. We display the correlation matrix in that the diagonal variable is 1. We got exact value between two variables.

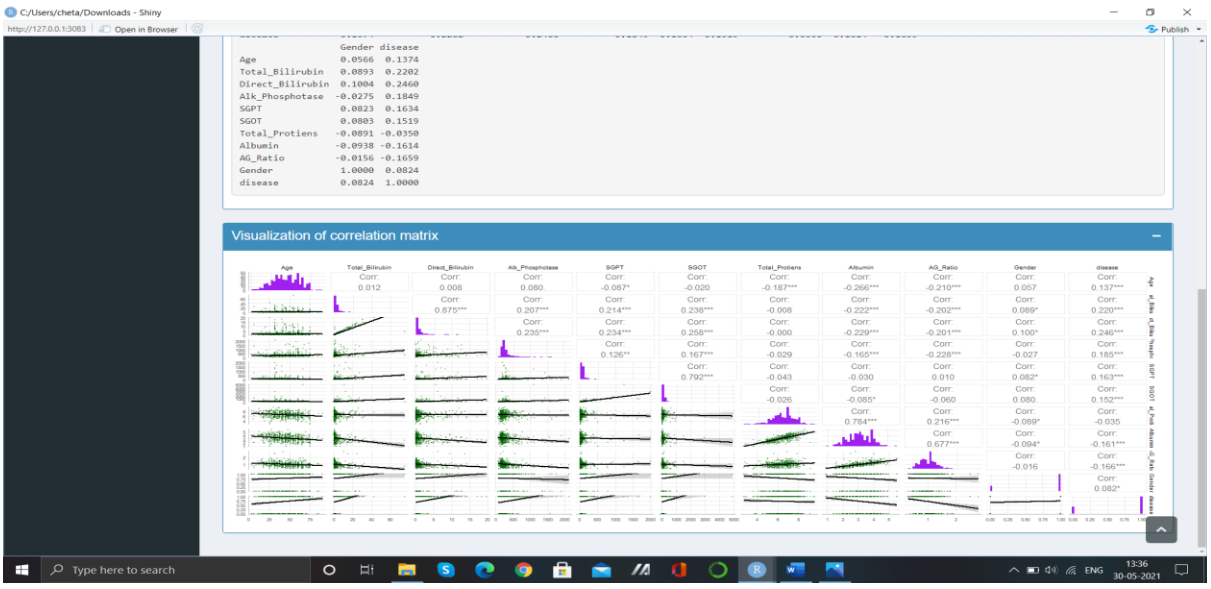


```
                    Age Total_Bilirubin Direct_Bilirubin Alk_Phosphotase    SGPT    SGOT Total_Protiens Albumin AG_Ratio
Age              1.0000          0.0118           0.0075          0.0804 -0.0869 -0.0199        -0.1875 -0.2659  -0.2098
Total_Bilirubin  0.0118          1.0000           0.8746          0.2067  0.2141  0.2378        -0.0081 -0.2223  -0.2021
Direct_Bilirubin 0.0075          0.8746           1.0000          0.2349  0.2339  0.2575        -0.0001 -0.2285  -0.2010
Alk_Phosphotase  0.0804          0.2067           0.2349          1.0000  0.1257  0.1672        -0.0285 -0.1655  -0.2282
SGPT            -0.0869          0.2141           0.2339          0.1257  1.0000  0.7920        -0.0425 -0.0297   0.0100
SGOT            -0.0199          0.2378           0.2575          0.1672  0.7920  1.0000        -0.0256 -0.0853  -0.0602
Total_Protiens  -0.1875         -0.0081          -0.0001         -0.0285 -0.0425 -0.0256         1.0000  0.7841   0.2157
Albumin         -0.2659         -0.2223          -0.2285         -0.1655 -0.0297 -0.0853         0.7841  1.0000   0.6767
AG_Ratio        -0.2098         -0.2021          -0.2010         -0.2282  0.0100 -0.0602         0.2157  0.6767   1.0000
Gender           0.0566          0.0893           0.1004         -0.0275  0.0823  0.0803        -0.0891 -0.0938  -0.0156
disease          0.1374          0.2202           0.2460          0.1849  0.1634  0.1519        -0.0350 -0.1614  -0.1659
                 Gender disease
Age              0.0566  0.1374
Total_Bilirubin  0.0893  0.2202
Direct_Bilirubin 0.1004  0.2460
Alk_Phosphotase -0.0275  0.1849
```

Figure 7.8 Window of Correlation matrix

Now correlation matrix as shown above, if we check correlation between two variables for that select column variable then select row variable check corresponding value of that variables. Eg we check Total_Protines vs Albumins is 0.78

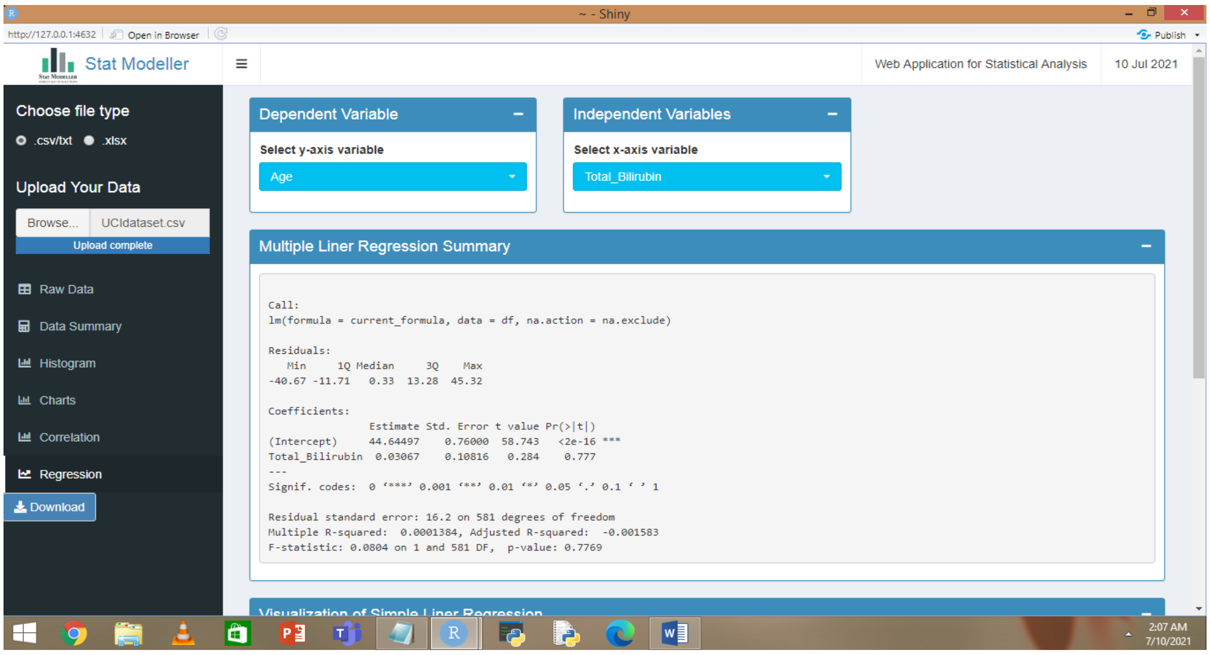
Step 7: To check visualization correlation matrix.



```
                 Gender disease
Age              0.0566  0.1374
Total_Bilirubin  0.0893  0.2202
Direct_Bilirubin 0.1004  0.2460
Alk_Phosphotase -0.0275  0.1849
SGPT             0.0823  0.1634
SGOT             0.0803  0.1519
Total_Protiens  -0.0891 -0.0350
Albumin         -0.0938 -0.1614
AG_Ratio        -0.0156 -0.1659
Gender           1.0000  0.0824
disease          0.0824  1.0000
```

Figure 7.9 Window of Correlation matrix Plot

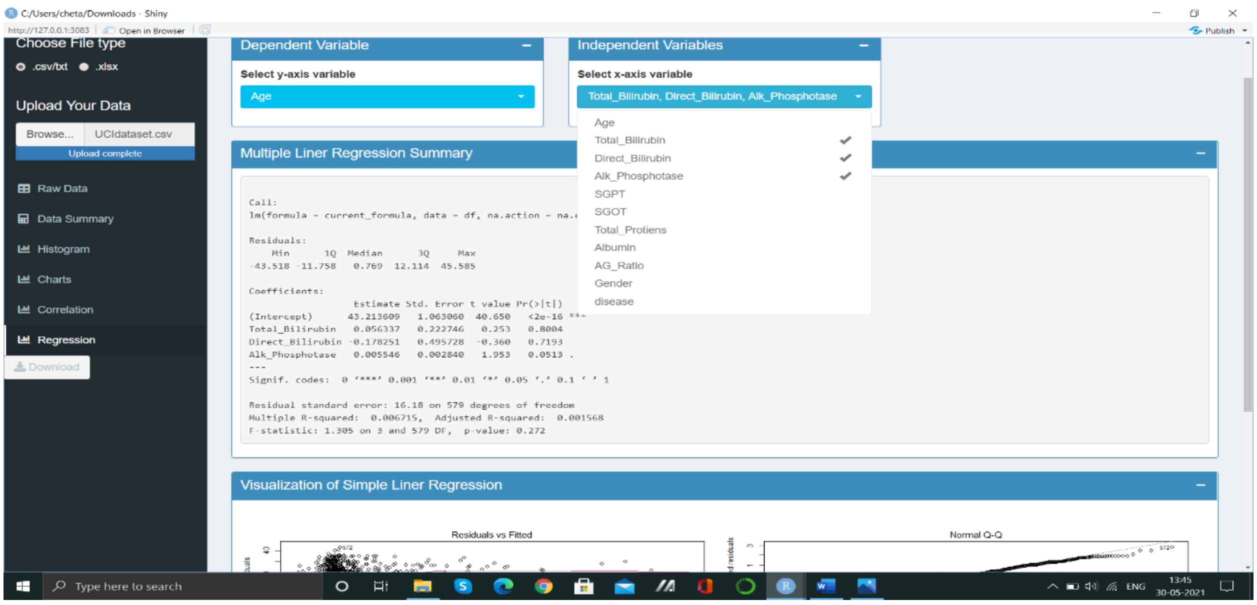
Step 8: The second last stage our application is simple linear regression. In simple linear regression we select y is the predicted value of the dependent variable (y) for any given value of the independent variable (x).



```
Call:
lm(formula = current_formula, data = df, na.action = na.exclude)

Residuals:
   Min     1Q Median     3Q    Max
-40.67 -11.71   0.33  13.28  45.32

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     44.64497    0.76000  58.743   <2e-16 ***
Total_Bilirubin  0.03067    0.10816   0.284    0.777
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16.2 on 581 degrees of freedom
Multiple R-squared:  0.0001384,	Adjusted R-squared:  -0.001583
F-statistic: 0.0804 on 1 and 581 DF,  p-value: 0.7769
```

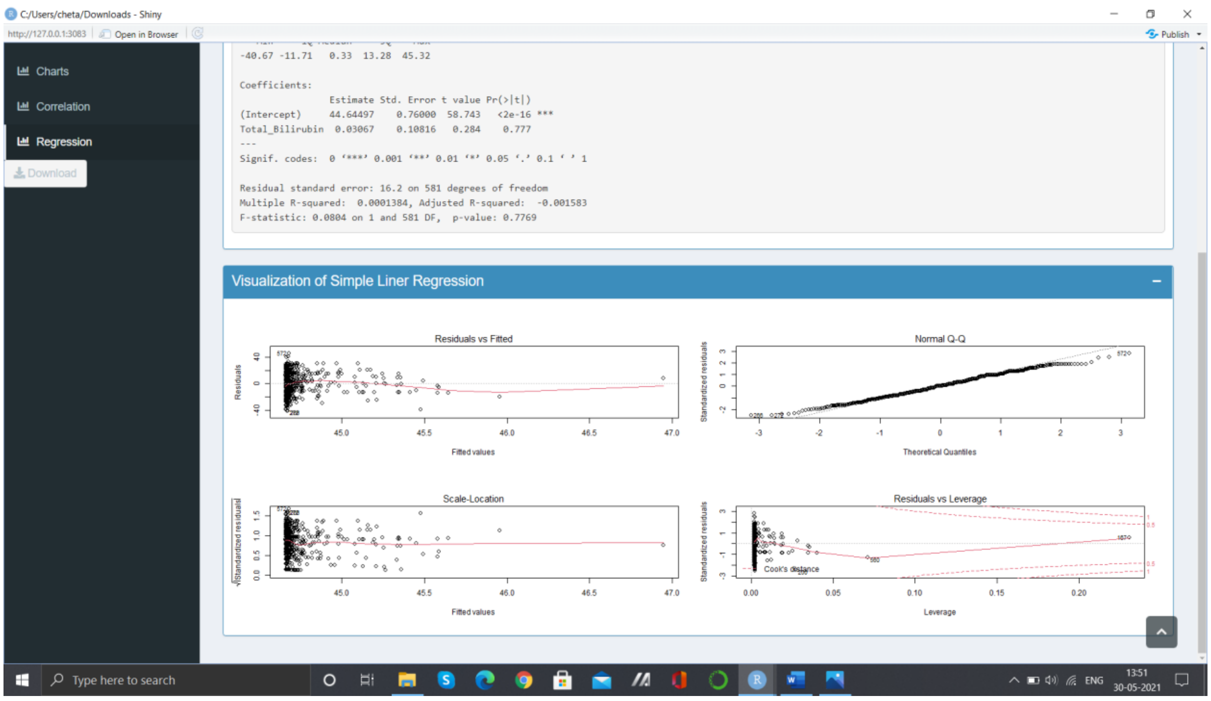
Figure 7.10 Window of Simple linear Regression

We add multiple variables in independent variables as well as dependent variables as multiple regression.



```
Call:
lm(formula = current_formula, data = df, na.action = na.e

Residuals:
    Min      1Q  Median      3Q     Max
-43.518 -11.758   0.769  12.114  45.585

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      43.213609   1.063060  40.650   <2e-16 ***
Total_Bilirubin   0.056337   0.222746   0.253   0.8004
Direct_Bilirubin -0.178251   0.495728  -0.360   0.7193
Alk_Phosphotase   0.005546   0.002840   1.953   0.0513 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16.18 on 579 degrees of freedom
Multiple R-squared:  0.006715,	Adjusted R-squared:  0.001568
F-statistic: 1.305 on 3 and 579 DF,  p-value: 0.272
```

Figure 7.11 Window of Multiple linear Regression



```
-40.67 -11.71   0.33  13.28  45.32

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     44.64497    0.76000  58.743   <2e-16 ***
Total_Bilirubin  0.03067    0.10816   0.284    0.777
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16.2 on 581 degrees of freedom
Multiple R-squared:  0.0001384, Adjusted R-squared:  -0.001583
F-statistic: 0.0804 on 1 and 581 DF,  p-value: 0.7769
```

Figure 7.12 Window of Visualization of Simple and Multiple linear Regression

CONCLUSION

The main objective of this project was to implement an application for facilitating the learning of data analysis techniques and basic statistical concepts. However, our application for statistical analysis, running smoothly is really where the rubber meets the road. This main objective has been successfully achieved, of course after many hours of developing code in R. Application is now a fully functional program, and has some of it most important functionalities implemented. There is still a long way to go, but this project has developed the main structure of application, and prepared the program to keep expanding and adding functionalities.

Application has been programmed using R, taking advantage of RStudio’s package shiny, and has been developed following a modular structure to facilitate the expansion, addition and modification of functionalities. A template to facilitate the programming of graphics has been created, and each one of the programmed R scripts includes explanations and comments to make the code easier to understand and modify.

The User Interface (UI) for each of the different options of application offers has been designed, always following the characteristics decided after the users’ needs analysis. The analysis of existing programs and the users’ needs study have been crucial to determine the features application has. Moreover, application UI is divided in three spaces: Menu (always visible on the left), Configuration Options of the selected functionality and a Results and Plots space. As application is intended to be interactive, it was decided that results would update automatically at every change on users’ inputs. Each one of these features was considered in the code, and thus the result is an app with a high level of interactivity and reactivity. During the design process, and taking into account all the analysis previously done, it was decided that the UI would be in a blue and white colour combination, that the Results box in all the functionalities pages would be on the right side, and that the Menu would be visible at all times (at the left side).. The structure and architecture of the app is programmed, but there is still a lot of work to do. There are functionalities still to implement and bugs to fix, and for sure changes to make based on users’ feedback and usage observation.