

# Learn Emotion Analysis with R

Perform Sentiment Assessments, Extract Emotions, and  
Learn NLP Techniques Using R and Shiny

PARTHA MAJUMDAR



## **Learn Emotion Analysis with R**

---

*Perform Sentiment Assessments,  
Extract Emotions, and Learn NLP  
Techniques Using R and Shiny*

---

**Partha Majumdar**



[www.bpbonline.com](http://www.bpbonline.com)

**FIRST EDITION 2021**

**Copyright © BPB Publications, India**

**ISBN: 978-93-90684-15-1**

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

#### **LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY**

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

**Distributors:**

## **BPB PUBLICATIONS**

20, Ansari Road, Darya Ganj

New Delhi-110002

Ph: 23254990/23254991

## **MICRO MEDIA**

Shop No. 5, Mahendra Chambers,

150 DN Rd. Next to Capital Cinema,

V.T. (C.S.T.) Station, MUMBAI-400 001

Ph: 22078296/22078297

## **DECCAN AGENCIES**

4-3-329, Bank Street,

Hyderabad-500195

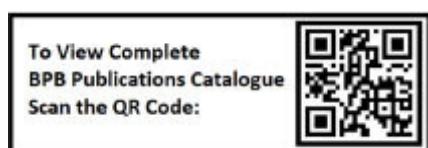
Ph: 24756967/24756400

## **BPB BOOK CENTRE**

376 Old Lajpat Rai Market,

Delhi-110006

Ph: 23861747



Published by Manish Jain for BPB Publications, 20 Ansari Road, Darya Ganj, New Delhi-110002 and Printed by him at Repro India Ltd, Mumbai

[www.bpbonline.com](http://www.bpbonline.com)

---

**Dedicated to**

*My beloved Wife:  
Deepshree Majumdar*

---

### ***About the Author***

**Partha Majumdar** is “*Just a* He has been involved in creating Enterprise Class Computer Software for various Ministries and Government Bodies in 7 Countries and for more than 20 Enterprises. He has been involved in creation of more than 10 Large Products and has been involved in more than 60 Projects around the Globe. He has been working in the Computer Software Industry for 32 years as of 2021.

### ***About the Reviewer***

**Simarpreet Singh** is a Data Science Consultant at one of the leading healthcare consulting company. He has 7+ years of experience in analysis, designing and implementing the Business Intelligence solutions to provide data driven insights. He has strong experience in business strategy and healthcare analytics to provide solutions by building statistical models using Machine Learning algorithms. In addition, he leverages data science techniques in order to grow business and have a positive impact on the key business KPIs.

## ***Acknowledgements***

There are a few people I want to thank for the continued and ongoing support they have given me during the writing of this book. First and foremost, I would like to thank my parents for continuously encouraging me for writing the book.

Then of course, Deepshree agreed to be the guinea pig. She is an Electronics Engineer; however, has no experience in programming. She agreed to run through all the programs in the book and this helped me test whether people with no prior background in programming could understand this book.

My gratitude to Arindam Mukhopadhyay who read all the contents and checked for all grammatical errors and consistence in usage of a fixed template. And Anita di went through all the contents and advised whether the sequence of the contents was logical and easy to understand.

I am grateful to the excellent online courses provided by top schools across the globe over the past few years. Few of them are: Data Science Specialisation (@Coursera by Johns Hopkins University), Machine learning (@Coursera by Mr. Andrew Ng from Stanford University), Deep Learning Specialisation (@Coursera by deeplearning.ai).

My gratitude also goes to the team at BPB Publication (including Saurabh and Anugraha) for being supportive enough to provide

me quite a long time to finish the book. Special mention is required for Mr. Simarpreet Singh who took the pains to conduct the Technical Review of the book.

## **Preface**

This book covers how to conduct Emotion Analysis based on a Lexicon. Emotion Analysis is very vital in the modern world and is a major stream of Natural Language Processing (NLP). It has many uses in Marketing, Human Resource Management, Medicine, etc. This book is based on practical work conducted on Emotion Analysis. Apart from discussing the concepts of Emotion Analysis, this book discusses how to develop applications for Emotion Analysis. The book contains complete code for the applications discussed in this book.

This book discusses Lexicon based Emotion Analysis as applications can be built if the Lexicon is available without any further requisite. The book discusses mechanisms to obtain the appropriate Lexicons. The book also discusses the contents of the Lexicons. Using these concepts, the readers can build their own Lexicons.

Another approach to Emotion Analysis is to use Machine Learning. However, this requires gathering appropriate data along with the labels based on which the machine can be trained. Conducting Emotion Analysis based on Machine Learning is not a major challenge if the concepts of Emotion Analysis and Machine Learning are known. So, using the concepts of Emotion Analysis from this book and juxtaposing knowledge of Machine Learning, the readers could take the next step for conducting Emotion Analysis based on Machine Learning. It needs noting that Emotion Analysis based on Machine Learning is not popular simply

because there is not sufficient data available based on which machines can be trained to conduct Emotion Analysis.

As all the programs discussed in this book are written in R, the book starts with discussing basics of R Programming. This is so that the readers do not have any issue in understanding the book even if they have no background in programming and/or R Programming. The applications discussed in this book are written in Shiny. So, the book discusses how to write Shiny Programs prior to discussing Emotion Analysis.

The book is divided into 4 sections.

Section 1 discusses R Programming. As R Programming is a huge subject and not the primary focus of this book, this book only discusses concepts and methods of R Programming required for understanding the programs for Emotion Analysis discussed in this book. All the concepts and methods of R Programming are discussed taking practical examples.

Section 2 discusses Shiny Programming. As Shiny Programming is a huge subject and not the primary focus of this book, this book only discusses concepts and methods of Shiny Programming required for understanding the application for Emotion Analysis discussed in this book. This section discusses 2 Shiny Applications. The complete code for these applications has been included in this section.

Section 3 discusses Emotion Analysis. Before discussing Emotion Analysis, the section discusses Sentiment Analysis. The section rounds off with discussion on building an application for Emotion Analysis using Shiny Programming using which Emotion Analysis can be conducted for any text available in a Text File. The book also discusses how to extend these concepts to conduct Emotion Analysis on text available in any other file format like PDF Files.

Section 4 discusses a practical application of Emotion Analysis. In this section, analysis of live data from Twitter is used for conducting Emotion Analysis. Reading this section, it will be possible to develop methods for obtaining data from Twitter and analysing data available from Twitter. The section rounds off with creating an application for analysis of Twitter Data.

The book includes a bonus chapter for discussing how to obtain data from WhatsApp and analyse the same.

This book consists of 12 chapters divided into 4 sections and a bonus chapter containing the following:

## SECTION 1

Chapter 1 will aim at introducing R. The chapter discusses how to obtain and install R and RStudio. The chapter discusses the requisites for Apple Mac machines and Windows machines. The chapter discusses how to test the installation. The chapter rounds off with discussion on what are vectors and packages.

[Chapter 2](#) will aim at introducing how to conduct simple operations using R Programming. The chapter introduces Data Frames which is one of the basic data structures in R. Various operations possible on Data Frames is discussed in this chapter. The chapter also discusses how to load data from CSV Files and from Databases. The chapter rounds off with creating visualisations from data available in Data Frames.

[Chapter 3](#) will aim at creating applications using R Programming. 2 applications are discussed in this chapter – Application to convert Number to Words and Application to create a Word Cloud from any given data (from Structured Data and from Unstructured Data). To be able to create the application, the chapter discusses how to write the basic programming constructs like conditions and loops using R Programming.

## SECTION 2

[Chapter 4](#) will aim at introducing Shiny Programming. The chapter discusses the basic structure of a Shiny Program. The chapter discusses how to create a Shiny Program, compile it, and publish it on the World Wide Web.

[Chapter 5](#) will discuss how to develop a Shiny Application for converting Numbers to Words.

[Chapter 6](#) will discuss how to develop a Shiny Application for creating a Word Cloud from Unstructured Data available in a Text File or a PDF File. This chapter also discusses what are

RMarkdowns and how to create RMarkdowns. The chapter discusses how to integrate Shiny Applications with RMarkdowns.

### SECTION 3

Chapter 7 will discuss Sentiment Analysis. The chapter starts with discussing what are Lexicons. The chapter discusses how to extract data from a given text for Sentiment Analysis. The chapter discusses how to conduct Sentiment Analysis and how to visualise the analysis.

Chapter 8 will discuss Emotion Analysis. The chapter starts with discussing how to clean the data for Emotion Analysis. Then, step by step analysis of the data is conducted leading to extracting the most prevalent emotion expressed in the text. The chapter also discusses how to visualise the analysis at every step.

Chapter 9 will discuss how to develop a Shiny Application for conducting Emotion Analysis of any text provided in the Text File.

### SECTION 4

Chapter 10 will introduce Twitter Data Analysis. This chapter discusses the preliminary requisites to be able to obtain and analyse data from Twitter.

Chapter 11 will discuss how to conduct Emotion Analysis on data obtained from Twitter. The chapter starts with discussing how to obtain data from Twitter. The chapter proceed to discuss analysis

of data obtained from Twitter. The chapter ends with discussion on how to extract the most prevalent emotion expressed in the fetched tweets.

[Chapter 12](#) will discuss how to develop a Shiny Application for conducting Emotion Analysis of Tweets.

[Bonus Chapter](#) will discuss how to obtain from WhatsApp and analyse it.

***Downloading the code***

***bundle and coloured images:***

Please follow the link to download the ***Code Bundle*** and the ***Coloured Images*** of the book:

<https://rebrand.ly/f4f79d>

***Errata***

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

[errata@bpbonline.com](mailto:errata@bpbonline.com)

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

---

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.bpbonline.com](http://www.bpbonline.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [business@bpbonline.com](mailto:business@bpbonline.com) for more details.

At you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

---

---

#### **BPB IS SEARCHING FOR AUTHORS LIKE YOU**

If you're interested in becoming an author for BPB, please visit [www.bpbonline.com](http://www.bpbonline.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

The code bundle for the book is also hosted on GitHub at In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at Check them out!

#### **PIRACY**

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at [business@bpbonline.com](mailto:business@bpbonline.com) with a link to the material.

#### **IF YOU ARE INTERESTED IN BECOMING AN AUTHOR**

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit

## REVIEWS

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit

---

## ***Table of Contents***

### **SECTION - 1**

#### **1. Getting Started with R**

Structure

Objectives

Brief introduction to R language

Setting up the R software

Obtaining the R software

Installing R

Invoking R

Setting up RStudio

Obtaining the RStudio software

Installing RStudio

Invoking RStudio

Introduction to packages

Installing packages

Installing packages using RStudio GUI

Installing packages using CLI

Loading libraries

Introduction to vector

Assigning vectors to variables

Checking the type of a vectors

Checking the length of a vectors

Conducting statistical operations on a vectors

Conclusion

Points to remember

Multiple choice questions

## Answers to MCQs

Questions

Key terms

## 2. Simple Operations Using R

Structure

Objectives

Introduction to data frame

Creating an empty data frame

Creating a Data Frame from vector(s).

Renaming column(s) in a data frame

Referencing row(s)/column(s) of a data frame

Referencing a data frame based on conditions

Adding column(s) to a data frame

Adding row(s) to a data frame

Sorting a data frame

Visualizing data in a data frame

Histogram

Box and Whisker chart

Pie chart

Scatter plots

Reading data from a CSV file

Reading data from a database

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## 3. Developing Simple Applications in R

Structure

Objectives

Programming conditions and loops in R

Functions in R

Application: number to words converter

Writing our own function

trim() function

helper() function

convert2DigitNumbers() function

convert3DigitNumbers() function

convertThousands() function

convertLakhs() function

convertCrores() function

number2wordsIndia() function

R library for number to words conversion

Application: Word Cloud

Creating a Word Cloud from structured data

Creating a Word Cloud from unstructured data

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## SECTION - 2

## 4. Structure of a Shiny Application

Structure

## Objectives

Creating a Shiny application using RStudio

Structure of a Shiny application

Client-side program

Displaying title

The body of the web page

Taking input

Displaying output

Server-side program

Running the application

Creating Separate Files for Client-side and Server-side Programs

ui.R

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## 5. Shiny Application 1

Structure

Objectives

Designing the user interface

The UI for SankhyaSaabdh

Coding the user interface

Displaying images in Shiny applications

Displaying text in Shiny applications

Accepting number only input in Shiny applications

Displaying the text output in Shiny applications

Creating layouts in Shiny applications

Coding the server side of SankhyaSaabdh

[The complete code of SankhyaSaabdh](#)

[Publishing a Shiny application](#)

[Conclusion](#)

[Points to remember](#)

[Multiple choice questions](#)

[Answers to MCQs](#)

[Questions](#)

[Key terms](#)

## **6. Shiny Application 2**

[Structure](#)

[Objectives](#)

[Creating RMarkdown document](#)

[Setting up the YAML header](#)

[Creating the body of the RMarkdown](#)

[Embedding R code in RMarkdown](#)

[Writing our First RMarkdown](#)

[Generating Output from RMarkdown](#)

[Developing Shiny application ShabdhMegh](#)

[What We will develop?](#)

[Creating tabs in Shiny applications](#)

[Creating radio button in Shiny applications](#)

[Inputting TEXT files in Shiny applications](#)

[Server-side Code side code for inputting text file](#)

[Inputting PDF files in Shiny applications](#)

[Server-side Code for inputting PDF Files](#)

[Switching Modes between TEXT file and PDF file](#)

[What we do before and after a file is input?](#)

[Creating output when an input file is provided](#)

[Computing the word frequency](#)

Creating the data table output  
Displaying the Word Cloud  
Displaying the progress bar  
Resetting output when a new input file is provided  
Programming the download button  
Programming the client-side of the Shiny application  
Programming the server-side of the Shiny application  
Programming the RMarkdown  
Complete code of ShabdMegh  
app.R  
ShabdMegh.RMD  
Conclusion  
Points to Remember  
Multiple choice questions  
Answers to MCQs  
Questions  
Key terms

### SECTION - 3

7. Sentiment Analysis  
Structure  
Objectives  
What is sentiment analysis?  
Approaches to sentiment analysis  
Knowledge-based approach  
Statistical approach  
Hybrid approach  
Sentiment knowledge bases in R  
afinn Lexicon

bing Lexicon

loughran Lexicon

nrc Lexicon

Conducting sentiment analysis

Extracting words from the text

Generating the words frequency

Finding sentiment across the text

Determine the sentiment of each word

Determine the sentiment of each line

Separate the counts of positive and negative sentiments

Calculate the net sentiments score for each chunk

Plot the net sentiments score

Determining the contribution of each word in the sentiment score

Visualizing words with positive and negative sentiment

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## 8. Emotion Analysis

Structure

Objectives

Conducting emotion analysis

Reading and cleaning data

Removing stop words

Word analysis

Check diversity of word frequencies

Most frequently used words in the text

Emotions expressed by the words – visualization 1

Emotions expressed by the words – visualization 2

Emotions expressed by the words – visualization 3

Sentiment flow in the text

Finding the most prevalent emotion in the text

Further analysis of the document

Bigram analysis

Trigram analysis

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## **9. ZEUSg**

Structure

Objectives

Design of ZEUSg

Landing page

Outputs from ZEUSg

Programming ZEUSg

Giving credit for using a library

Libraries used to develop ZEUSg

Creating tabs within a tab

Reading the input file

The complete code of ZEUSg

app.R

Conclusion

Points to remember

Multiple choice questions

[Answers to MCQs](#)

[Questions](#)

[Key terms](#)

## **SECTION - 4**

### **10. Introduction to Twitter Data Analysis**

[Structure](#)

[Objectives](#)

[Introduction to Twitter](#)

[Twitter Developer Account](#)

[URL for creating a Twitter Developer Account](#)

[Creating a Twitter Developer Account](#)

[Steps for creating an app](#)

[App approval by Twitter](#)

[Google Maps API key](#)

[URL for creating a Google Maps API key](#)

[Conclusion](#)

[Points to remember](#)

[Multiple choice questions](#)

[Answers to MCQs](#)

[Questions](#)

[Key terms](#)

### **11. Emotion Analysis on Twitter Data**

[Structure](#)

[Objectives](#)

[A Few Features of Twitter](#)

[Fetching data from Twitter](#)

[Library required for fetching tweets](#)

Setting up a Twitter account

Authenticating the Twitter account

Fetching tweets by search string(s).

Fetching tweets for a Location

Displaying tweets using DT

Displaying tweets with their links

Analyzing information obtained from Twitter

Time Series chart for when tweets were posted

Country analysis

Place analysis

Language analysis

User analysis

Source analysis

Hashtag analysis

Location analysis with maps

Interactive maps using a leaflet

Emotion analysis of tweets

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## 12. Chidiya

Structure

Objectives

Features of Chidiya

Landing page

Global code

Loading the required libraries

Global functions

Generic function to display a message box

Function to fetch the subjects of tweets stored in the Chidiya database

Function to fetch the Previously Searched Subjects

Function to read the initial set of tweets

Function to extract words from tweets

Function to extract emotions from tweets

Function to save statistics

Initialization code

Reading the system parameters

Setting up the Twitter Account

Setting up the database

Setting up the Google Maps API

Setting up the Lexicon

Setting up the data for initial display

Server-side code

Initiating response to user input

Fetching the tweets

Making Chidiya responsive

Client-side code

The complete for code Chidiya

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

**Bonus-WhatsApp Chat Analysis**

Structure

## Objectives

Introduction to WhatsApp

How to download data from WhatsApp?

Downloading WhatsApp data from Apple iPhones

Downloading WhatsApp data from Android phones

The structure of WhatsApp chat data

Reading WhatsApp chat data using R

Visualizing WhatsApp chat data

Messages per day

Messages per weekday

Radar chart

Messages per hour

Heat map

Messages per author

Emoji analysis

Word analysis

Conclusion

Points to remember

Multiple choice questions

Answers to MCQs

Questions

Key terms

## Index

## SECTION - 1

## CHAPTER 1

### Getting Started with R

R is a free software environment for statistical computing and graphics. R language R language is very widely used. Ever since Data Sciences have gained popularity, the use of R language has increased tremendously. It is used by scientists, researchers, students, and across all industry. R language is open-source and thus, many developers contribute to its growth. As a result, R language has a very rich library for almost any aspect of developing systems for almost all requirements across almost all domains.

We will use R language to create our system for emotion analysis in this book. So, let us discuss the basics of R language for you to understand the programs discussed in this book.

## Structure

In this chapter, we will discuss the following topics:

Brief introduction to R language

Setting up the R software

- Obtaining the R software
- Installing R
- Invoking R

Setting up RStudio

- Obtaining the RStudio software
- Installing RStudio
- Invoking RStudio

Introduction to packages

Introduction to vector



## Objectives

After studying this unit, you should be able to:

Install R and RStudio

Install packages and load libraries

Perform statistical operations using R

Create graphs using R

## Brief introduction to R language

R language was initially created by *Ross Ihaka* and *Robert Gentleman* at the *University of New Zealand*. The project was conceived in 1992. An initial version was released in 1995. The first stable version was released on February 2000, versioned as 1.0. The latest version (at the time of writing this book) is R version 4.0.0 which was released on April 2020.

R language originated from S language developed by *John Chambers* at the Bell Labs. R is an implementation of the S language developed using C, and R itself.

R is supported by *The R Project for Statistical Computing*. The official website is <http://www.R-project.org>. Since the mid 1997, there has been a core group (the *Core Team*) responsible for modifying the R source code archive.

The Comprehensive R Archive Network (CRAN) is a collection of sites which carry identical materials consisting of the R distribution(s), the contributed extensions, the documentation for R, and binaries. The official website of CRAN is <http://cran.r-project.org>. Its head office is in Austria and has many mirror sites around the world. All materials related to R language can be obtained from one of the CRAN sites.

One must read the documentation provided at the following URL:  
There are some interesting facts including the names of the

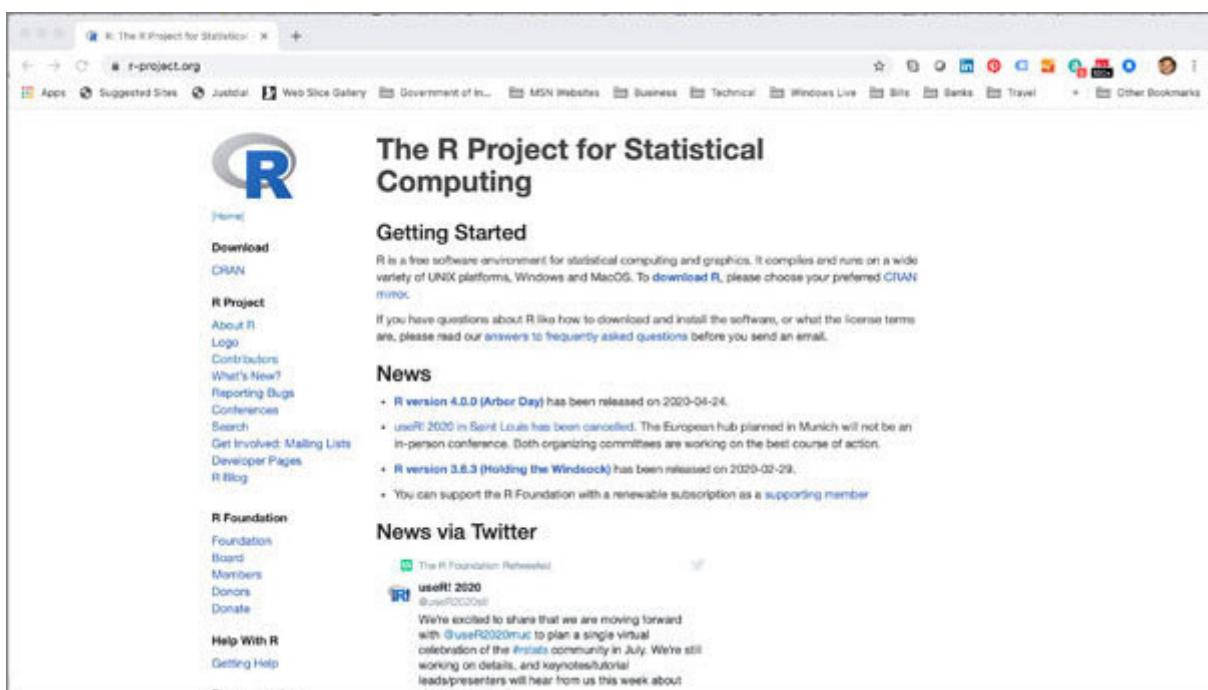
people who can modify the R Core software, why R was named R, what is *R* what is *S*, what is S-Plus, what is R-Plus, how I can create an R package, how I can contribute to R, how to report R bugs, etc.

## Setting up the R software

R can be installed on any UNIX-like machine, Apple Mac, and Windows machines. We will walk through the process of installing R on an Apple Mac.

## Obtaining the R software

To obtain the R Software, visit On invoking this URL on Google Chrome, the web page looks like as shown below:



**Figure 1.1: The R Project Home Page**

On this web page, click on the link to CRAN site as shown in the following screenshot:

The R Project for Statistical Computing

**Getting Started**

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and Mac OS. To [download R](#), please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked [questions](#) before you send an email.

**News**

- R version 4.0.0 (Arbor Day) has been released on 2020-04-24.
- useR! 2020 in Saint Louis has been canceled. The European hub planned in Munich will not be an in-person conference. Both organizing committees are working on the best course of action.
- R version 3.6.3 (Holding the Windsock) has been released on 2020-03-29.
- You can support the R Foundation with a renewable subscription as a [supporting member](#).

**News via Twitter**

The R Foundation Retweeted  
useR! 2020 @useR2020org We're excited to share that we are moving forward with useR2020muc to plan a single virtual celebration of the #rstats community in July. We're still working on details, and keynotes/tutorial leads/presenters will hear from us this week about where we stand.

**Figure 1.2:** Click on the CRAN link to reach a CRAN mirror

On clicking the CRAN link, you should get a web page as shown below with links to all the Mirror sites:

CRAN Mirrors

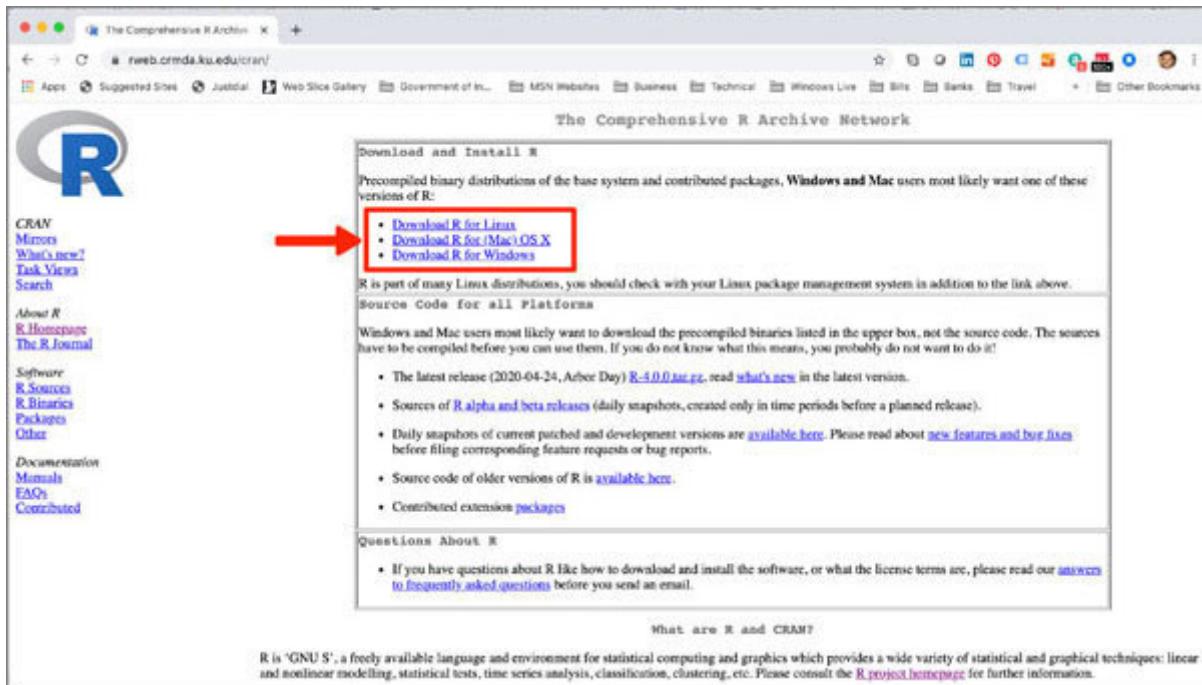
The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#), [windows release](#), [windows old release](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud	Automatic redirection to servers worldwide, currently sponsored by RStudio
<a href="https://cloud.r-project.org/">https://cloud.r-project.org/</a>	
Algeria	<a href="https://cran.usdhb.dz/">https://cran.usdhb.dz/</a>
Argentina	<a href="http://mirror.fcaglp.unlp.edu.ar/CRAN/">http://mirror.fcaglp.unlp.edu.ar/CRAN/</a>
Australia	<a href="https://cran.csiro.au/">https://cran.csiro.au/</a> <a href="https://mirror.su.oz.ac.au/pub/CRAN/">https://mirror.su.oz.ac.au/pub/CRAN/</a> <a href="https://cran.ms.unimelb.edu.au/">https://cran.ms.unimelb.edu.au/</a> <a href="https://cran.curtin.edu.au/">https://cran.curtin.edu.au/</a>
Austria	<a href="https://cran.wu.ac.at/">https://cran.wu.ac.at/</a>
Belgium	<a href="https://www.freestatistics.org/cran/">https://www.freestatistics.org/cran/</a> <a href="https://lib.ugent.be/CRAN/">https://lib.ugent.be/CRAN/</a>
Brazil	<a href="https://mbechb.usc.br/mirrors/cran/">https://mbechb.usc.br/mirrors/cran/</a> <a href="https://cran.r.c3sl.ufpr.br/">https://cran.r.c3sl.ufpr.br/</a> <a href="https://cran.hccug.be/">https://cran.hccug.be/</a> <a href="https://vms.fim.vtaps.br/CRAN/">https://vms.fim.vtaps.br/CRAN/</a> <a href="https://bciegr.resslab.mpg.de/CRAN/">https://bciegr.resslab.mpg.de/CRAN/</a>
Bulgaria	<a href="https://ftp.uni-sofia.bg/CRAN/">https://ftp.uni-sofia.bg/CRAN/</a>
Canada	<a href="http://mirror.its.sfu.ca/mirror/CRAN/">http://mirror.its.sfu.ca/mirror/CRAN/</a>
Europe	<a href="https://cran.r-project.org/mirror/europ/">https://cran.r-project.org/mirror/europ/</a>
France	<a href="https://cran.math.sciences.univ-lyon1.fr/">https://cran.math.sciences.univ-lyon1.fr/</a>
Germany	<a href="https://cran.r-project.org/mirror/de/">https://cran.r-project.org/mirror/de/</a>
India	<a href="https://cran.r-project.org/mirror/in/">https://cran.r-project.org/mirror/in/</a>
Ireland	<a href="https://cran.r-project.org/mirror/ie/">https://cran.r-project.org/mirror/ie/</a>
Italy	<a href="https://cran.r-project.org/mirror/it/">https://cran.r-project.org/mirror/it/</a>
Japan	<a href="https://cran.r-project.org/mirror/ja/">https://cran.r-project.org/mirror/ja/</a>
Korea	<a href="https://cran.r-project.org/mirror/kr/">https://cran.r-project.org/mirror/kr/</a>
Netherlands	<a href="https://cran.r-project.org/mirror/nl/">https://cran.r-project.org/mirror/nl/</a>
Poland	<a href="https://cran.r-project.org/mirror/pl/">https://cran.r-project.org/mirror/pl/</a>
Portugal	<a href="https://cran.r-project.org/mirror/pt/">https://cran.r-project.org/mirror/pt/</a>
Russia	<a href="https://cran.r-project.org/mirror/rus/">https://cran.r-project.org/mirror/rus/</a>
Singapore	<a href="https://cran.r-project.org/mirror/sing/">https://cran.r-project.org/mirror/sing/</a>
Spain	<a href="https://cran.r-project.org/mirror/es/">https://cran.r-project.org/mirror/es/</a>
Sweden	<a href="https://cran.r-project.org/mirror/se/">https://cran.r-project.org/mirror/se/</a>
Switzerland	<a href="https://cran.r-project.org/mirror/ch/">https://cran.r-project.org/mirror/ch/</a>
United Kingdom	<a href="https://cran.r-project.org/mirror/uk/">https://cran.r-project.org/mirror/uk/</a>
United States	<a href="https://cran.r-project.org/mirror/us/">https://cran.r-project.org/mirror/us/</a>

**Figure 1.3:** Links to CRAN mirrors

Click on any of the mirror sites to obtain the following web page:

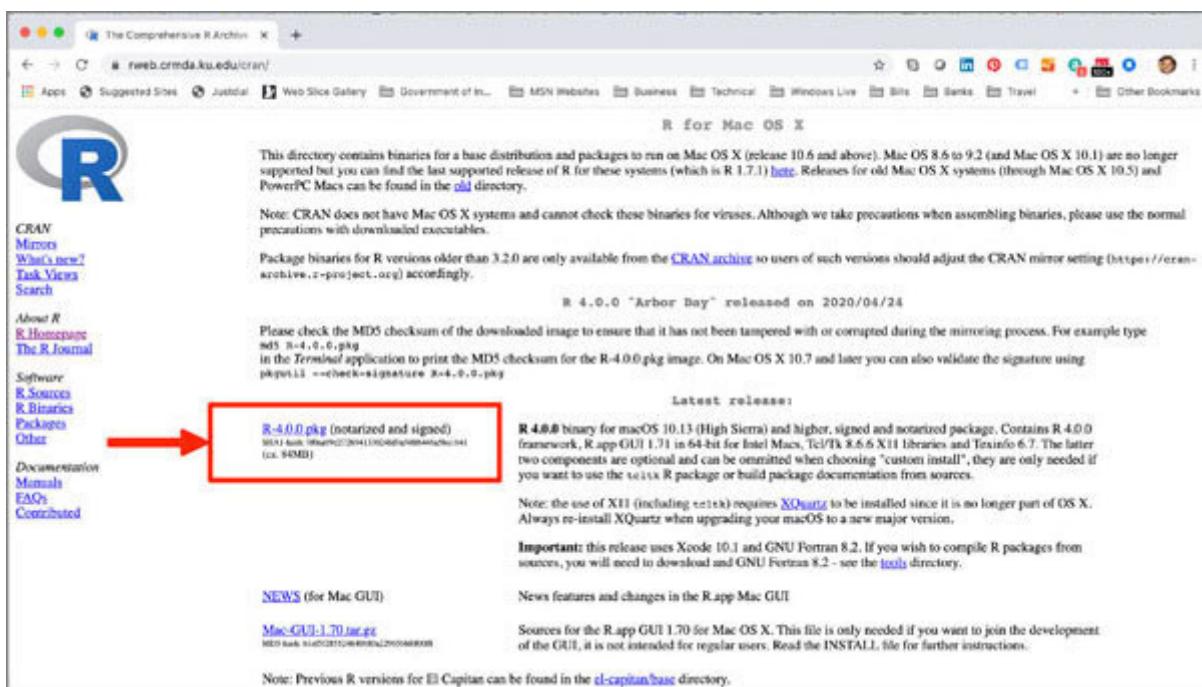


**Figure 1.4:** R Software download web page

In the box highlighted in RED, you will find the links to the R software for different operating systems. Click on the appropriate link as per the operating system of your machine to obtain the R software.

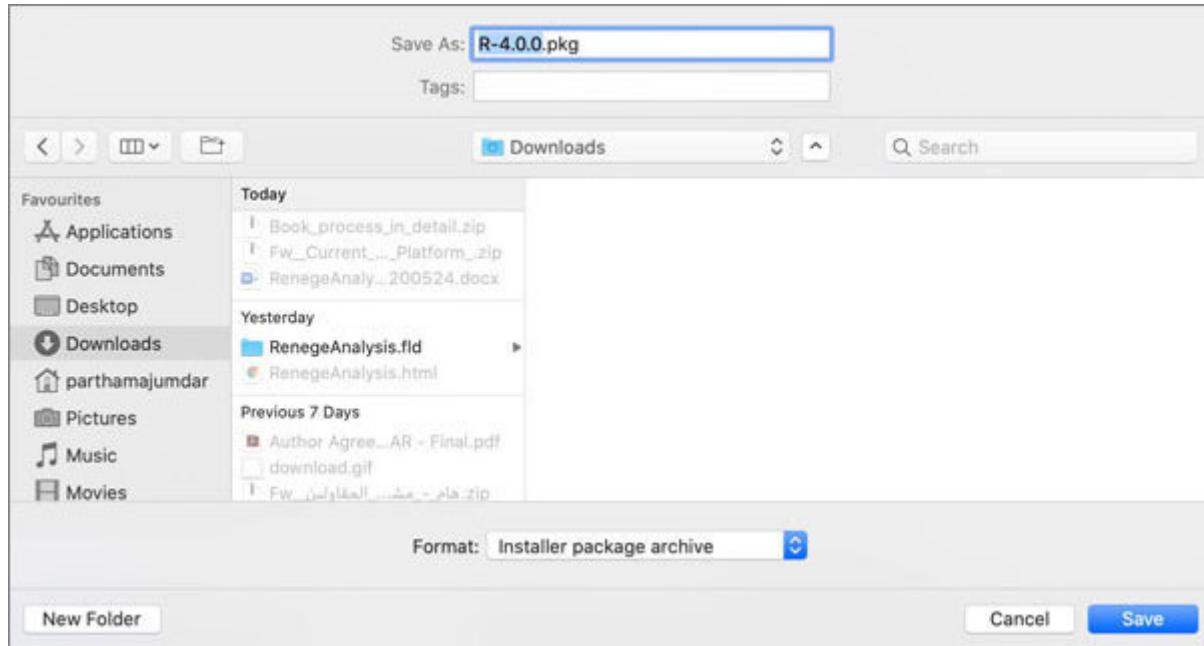
## Installing R

On the web page web page shown in *figure* click on the link **Download R for (Mac) OS** You should get this web page as shown in the following screenshot:



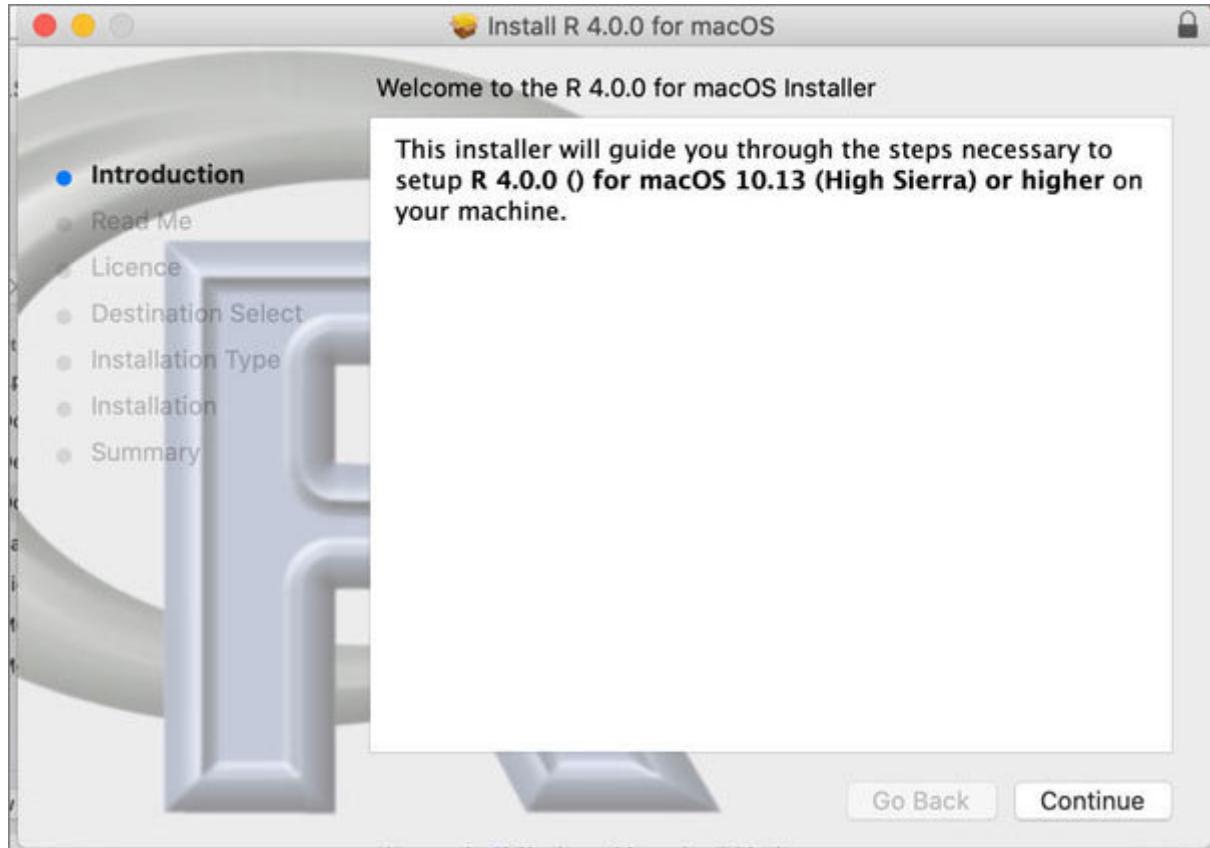
*Figure 1.5: R software download link for Mac*

Click on the link as highlighted in the RED-colored color box. A dialog box asking for the location to download the R software package is displayed. Select a folder and click



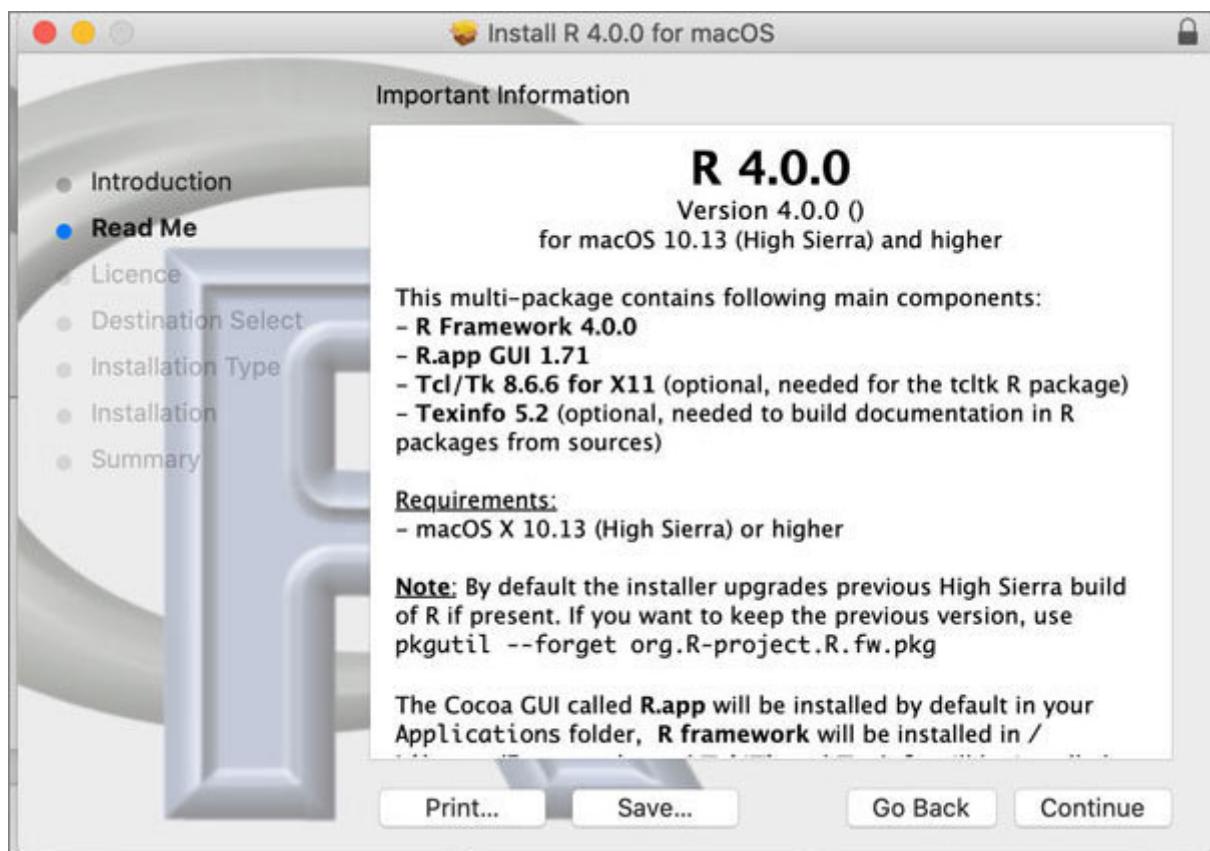
**Figure 1.6:** R Package download dialog box

Once the software has been downloaded, locate the downloaded package and run it. When the package is run, the installation process starts with this dialog box as shown in the following screenshot:



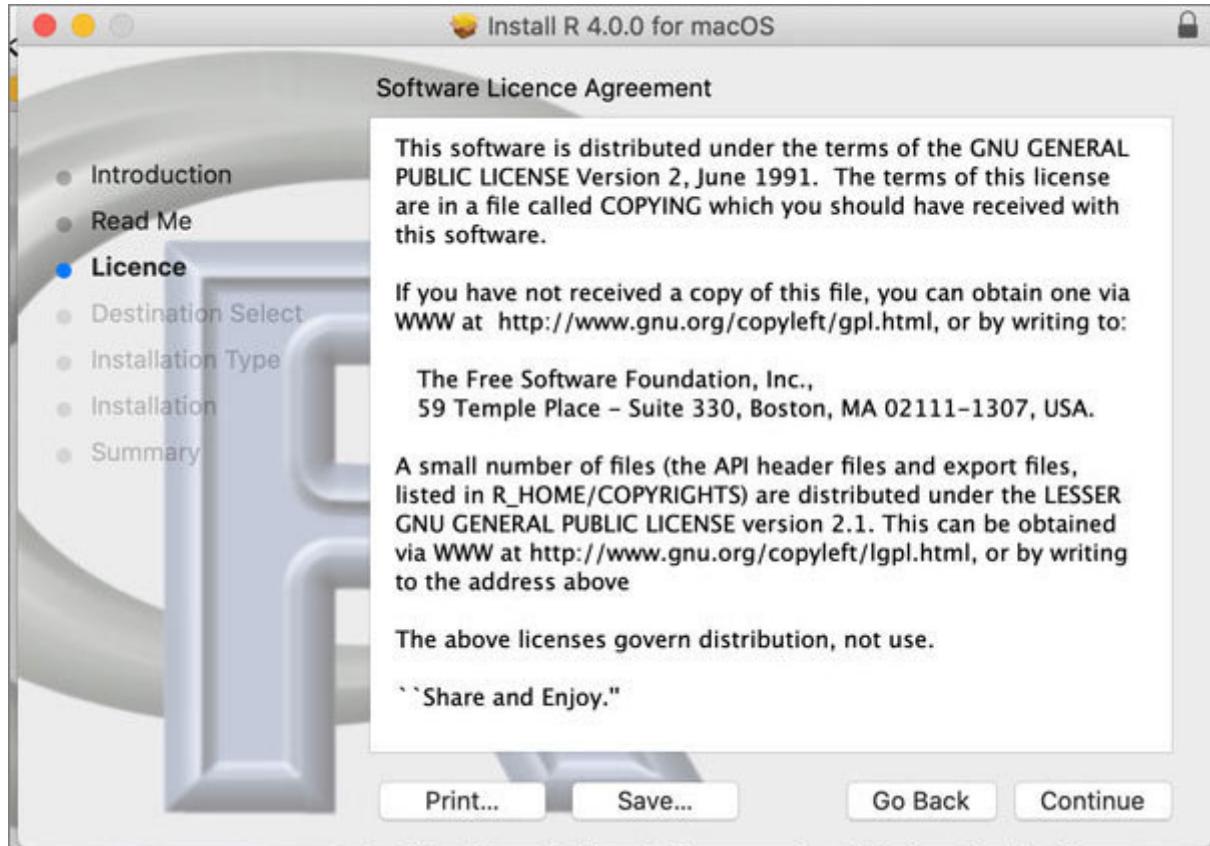
**Figure 1.7:** R Installation step 1

Click on the **Continue** button. The dialog box changes as shown in the following screenshot:



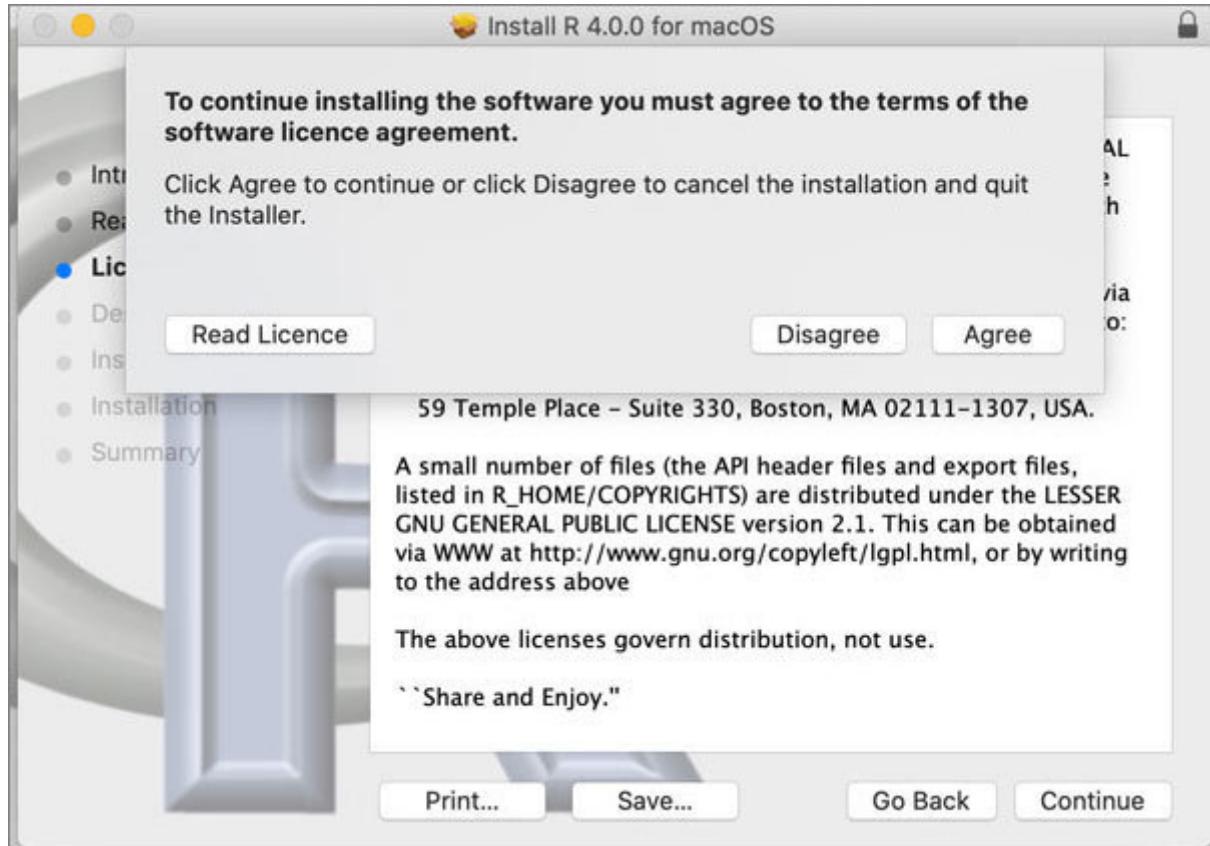
**Figure 1.8: R Installation step 2**

Read the information provided. You can print it and/or save it to your disk. Once done, click the **Continue** button. The dialog box shown in the following screenshot:



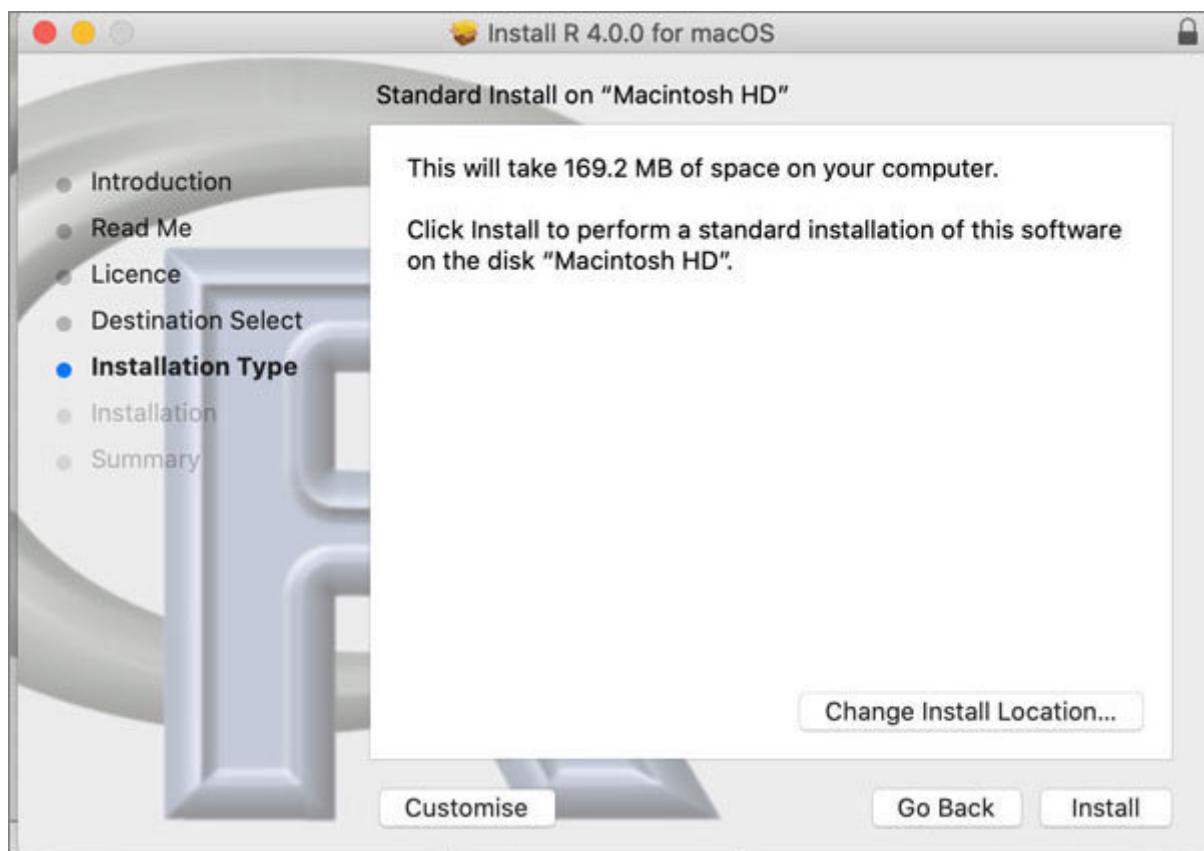
**Figure 1.9:** R installation step 3

This is the licensing agreement. R is free software. Read the information provided. You can print it and/or save it in your computer. Once done, click the **Continue** button. The dialog box shown in the following screenshot:



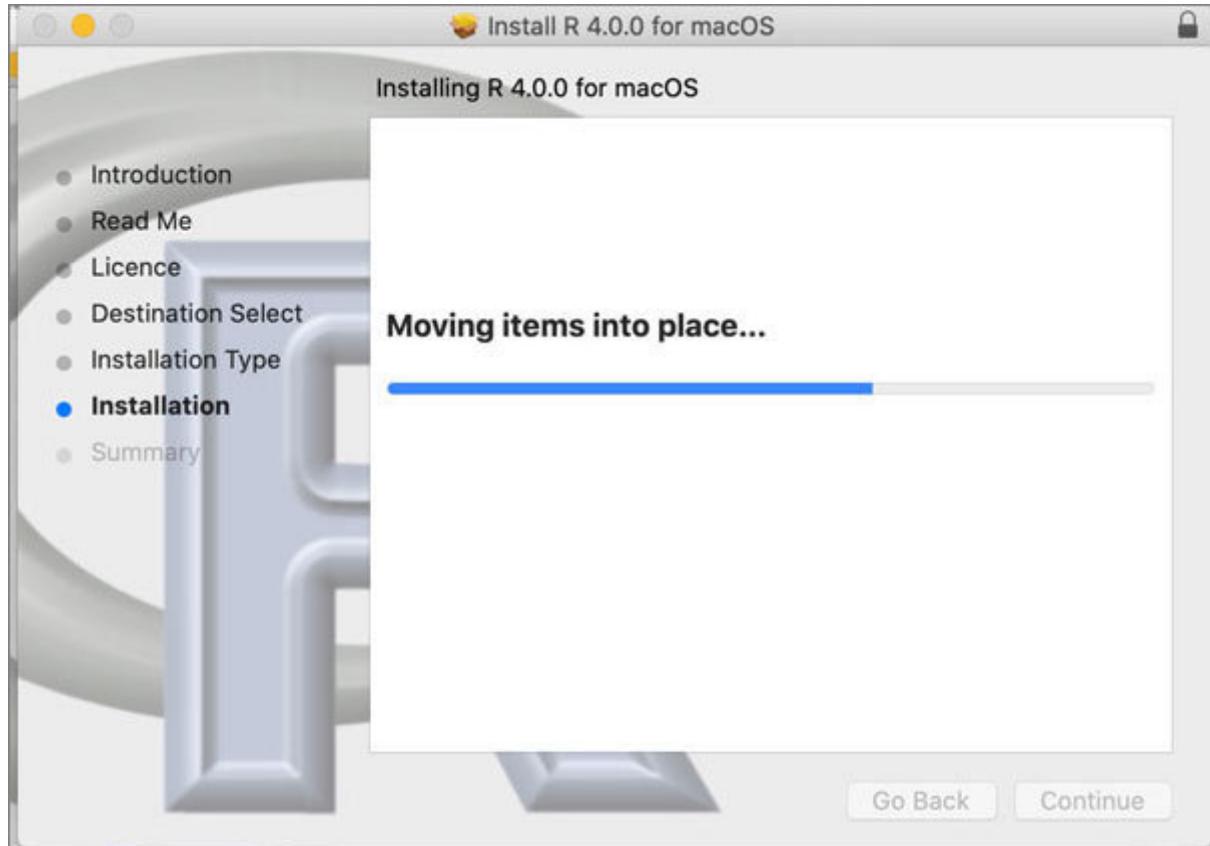
**Figure 1.10:** R installation step 4

Click the **Agree** button to continue. The following dialog box in the following screenshot:



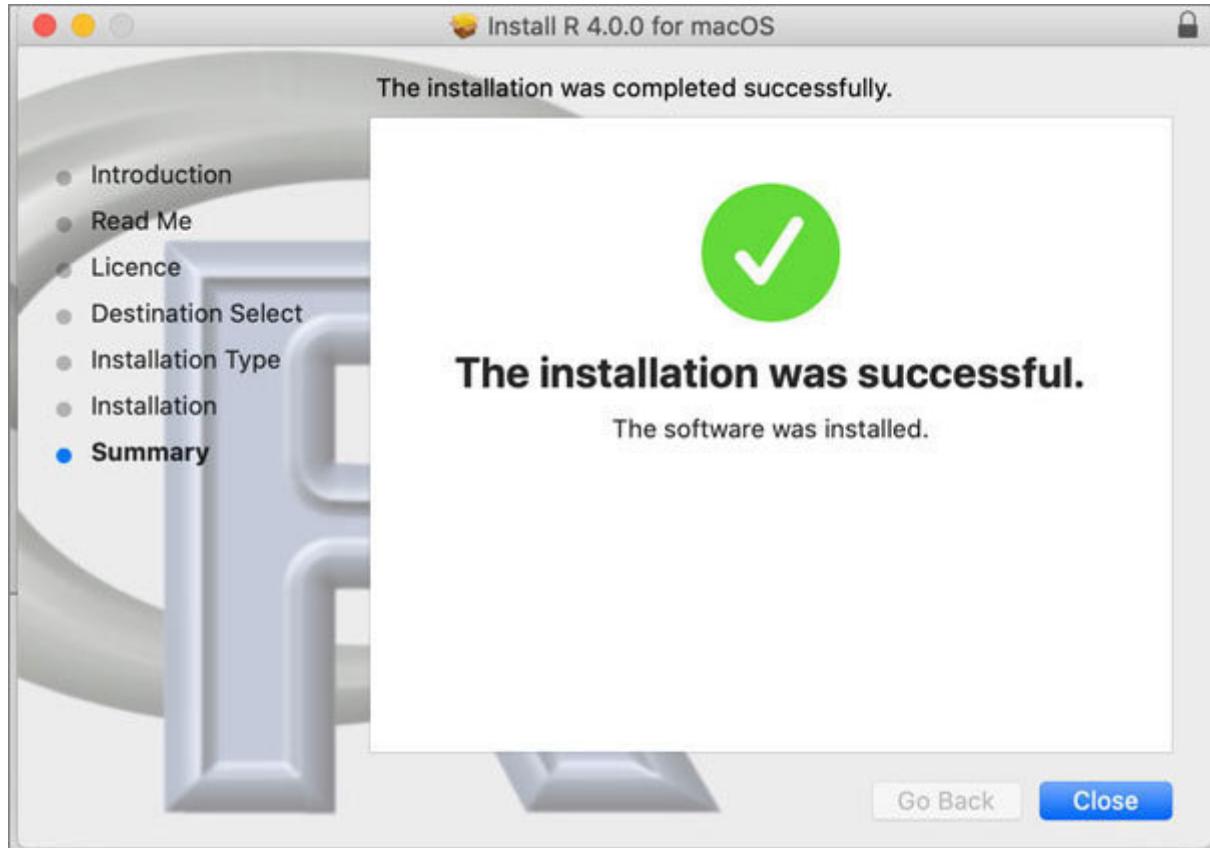
**Figure 1.11:** *R* installation step 5

Select the destination disk where R needs installing and click The installation starts after that:



**Figure 1.12:** R installation step 6

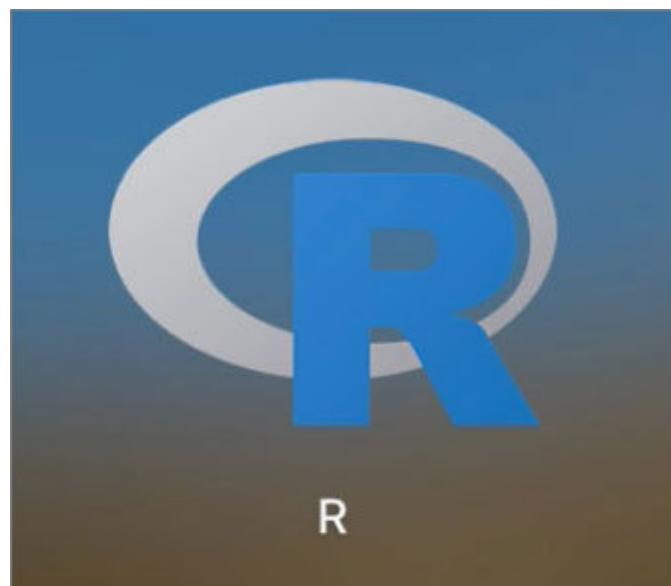
When the installation is complete, a message, as shown in the following screenshot:



*Figure 1.13:* R installation step 7

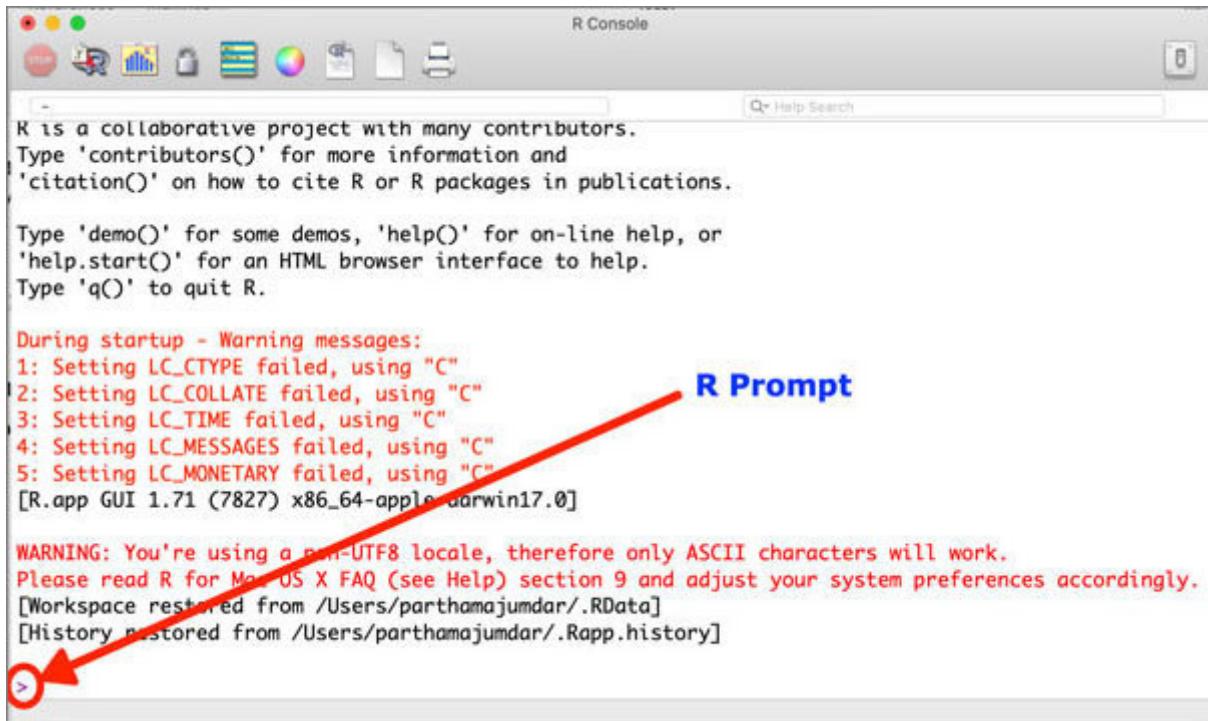
## [Invoking R](#)

Once R is installed, click on **Launchpad** to locate the R software. The icon for the R Software is as shown in the following screenshot:



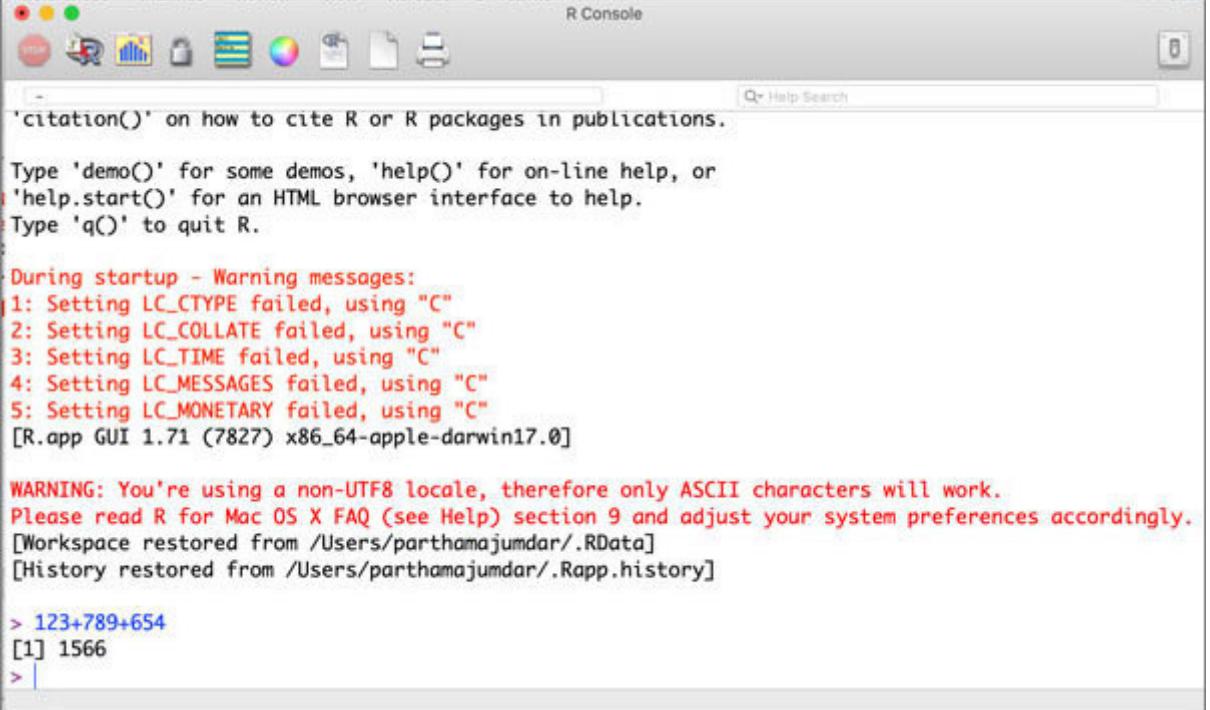
**Figure 1.14:** Click this icon to invoke the R software

On clicking this icon, the R software is invoked. The **R Console** looks like as shown in the following screenshot:



**Figure 1.15: R Console**

The > symbol at the bottom of the screen is R Prompt. Any R command needs to be provided at the R prompt. Try typing any mathematical formula in front of the R Prompt and press You should see the result. An illustration is provided in the following screenshot:



The screenshot shows the R Console window on a Mac OS X desktop. The window title is "R Console". The menu bar includes "File", "Edit", "View", "Help", and "Search". The main pane displays the R startup message, warning about non-UTF8 locale, workspace restoration, and history restoration. A command line input "123+789+654" is entered, followed by its result "[1] 1566".

```
R Console
citation() on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
[R.app GUI 1.71 (7827) x86_64-apple-darwin17.0]

WARNING: You're using a non-UTF8 locale, therefore only ASCII characters will work.
Please read R for Mac OS X FAQ (see Help) section 9 and adjust your system preferences accordingly.
[Workspace restored from /Users/parthamajumdar/.RData]
[History restored from /Users/parthamajumdar/.Rapp.history]

> 123+789+654
[1] 1566
> |
```

**Figure 1.16:** Performing simple arithmetic using R

The process to install R Software on Windows and Linux machine is very similar. So, I will not repeat the process. You need to take care that the file types in Windows and Linux is different from that in Apple Mac. Also, the method of invoking an application in Windows and Linux is different from that in Apple Mac.

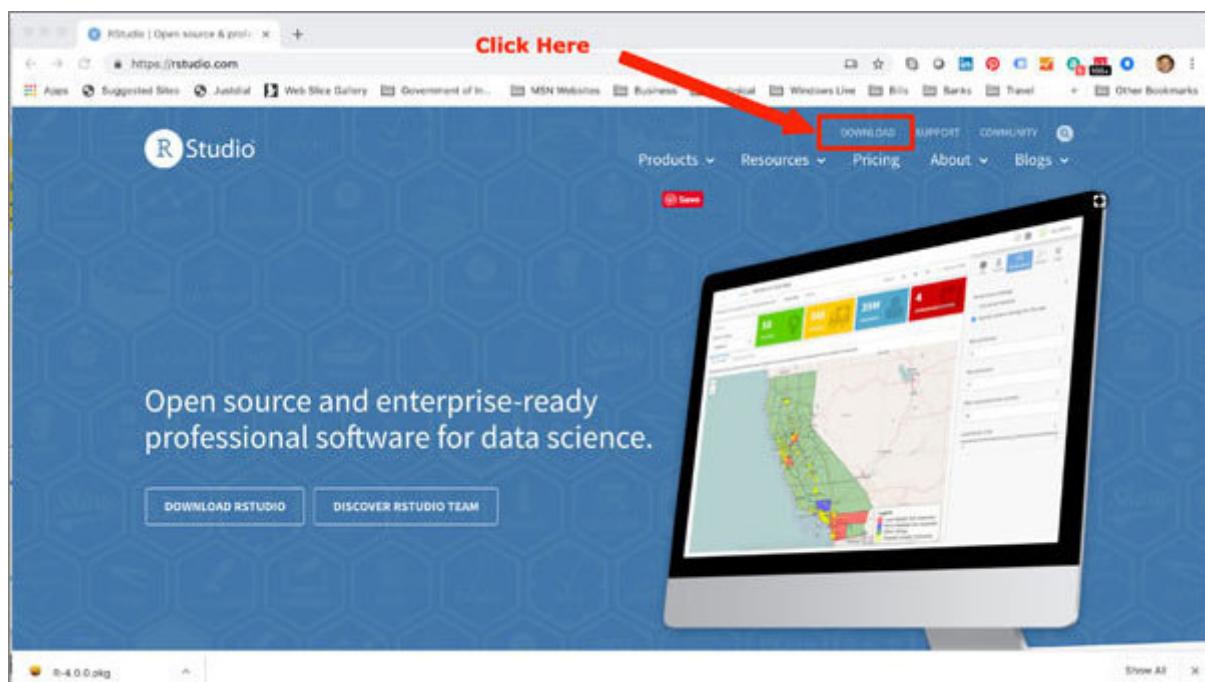
## Setting up RStudio

RStudio is an **Integrated Development Environment (IDE)** for R. Initially, RStudio was exclusively for R programming. However, it also supports Python programming of late. An IDE provides all requisite tools for programming. RStudio provides templates for creating, running, and debugging the different types of R programs like R Scripts, R Markdowns, Shiny Applications, etc. Besides, RStudio provides a Shell Interface using which commands can be run on the Shell of the operating system. It provides **Graphical User Interface** for conducting activities like installing/updating libraries, setting working directories, debugging programs, version control of code units, etc.

Other IDEs are also available for R Programming, like Jupyter.

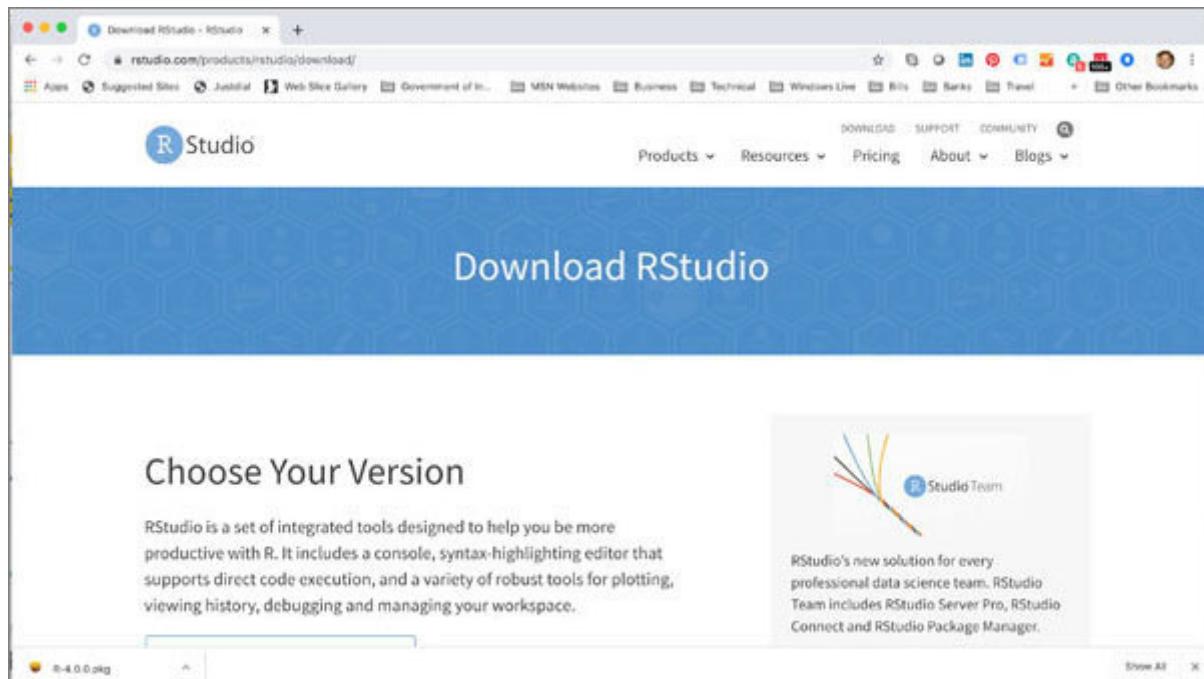
## Obtaining the RStudio software

To obtain the RStudio software, visit [On invoking this URL on Google Chrome, the web page looks like as shown in the following screenshot:](https://rstudio.com)



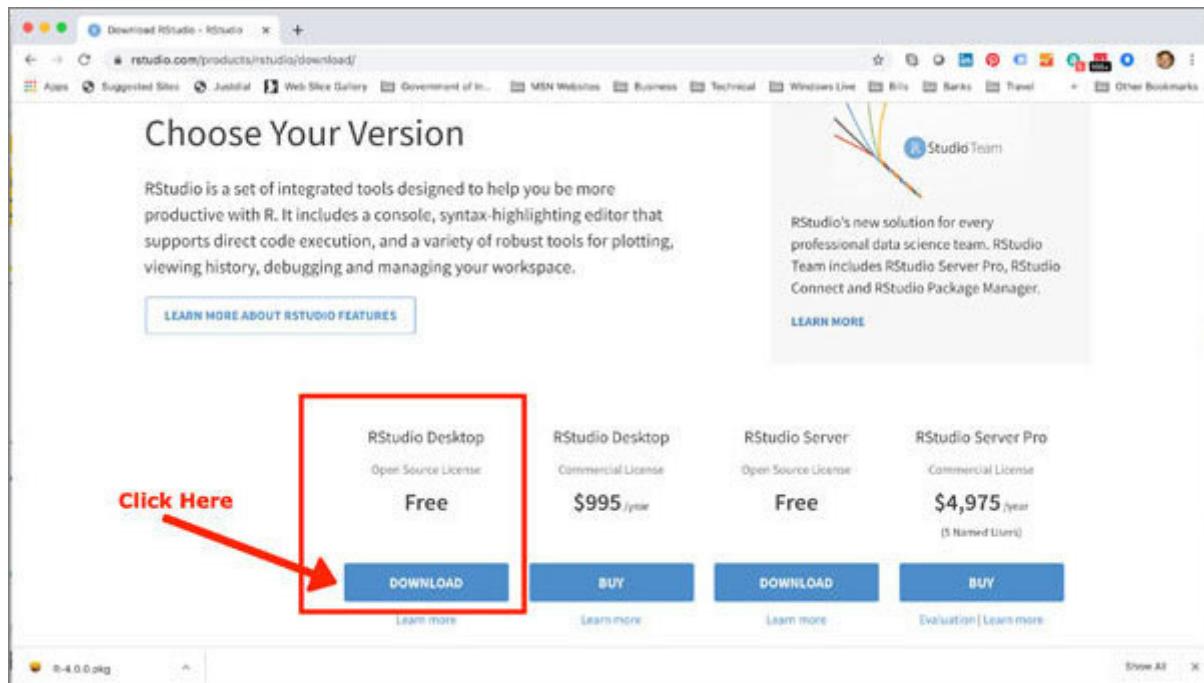
**Figure 1.17:** RStudio home page

Click on the **Download** link as shown in [figure](#). The following web page in the following screenshot:



**Figure 1.18:** RStudio download page

Scroll down on this page to get the download options as shown in the following screenshot:

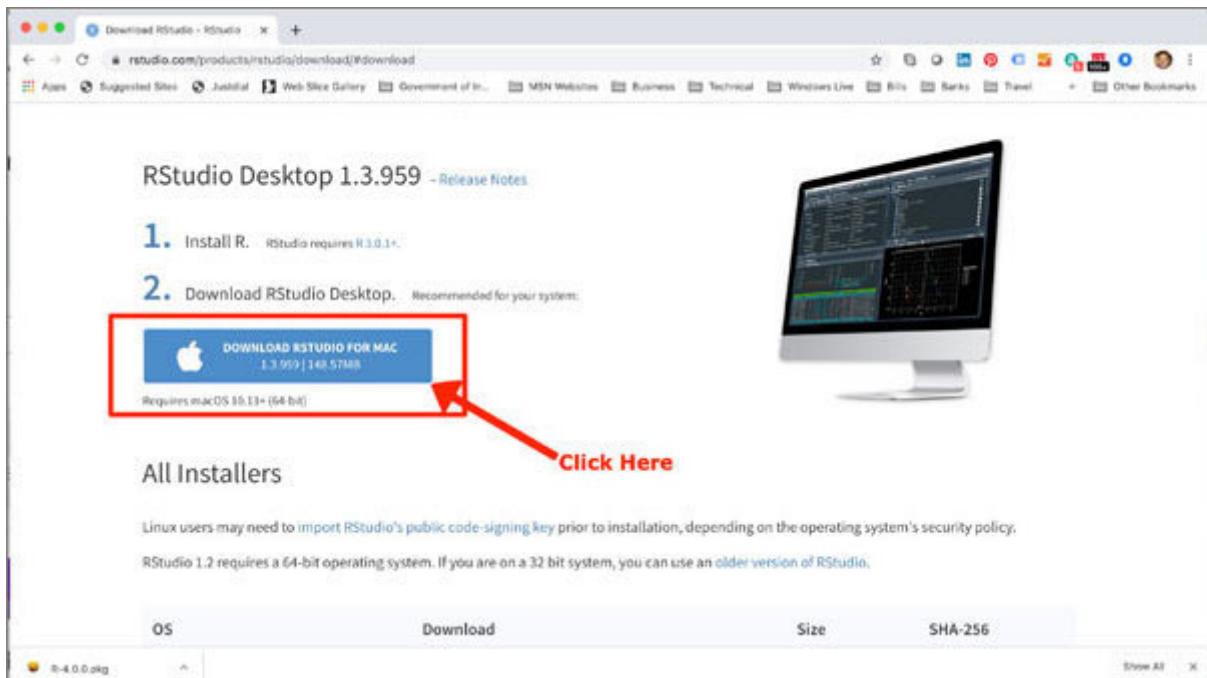


***Figure 1.19: RStudio download options***

**Select the download option Download option suitable for you. The RStudio Desktop version which is Free is enough for most requirements.**

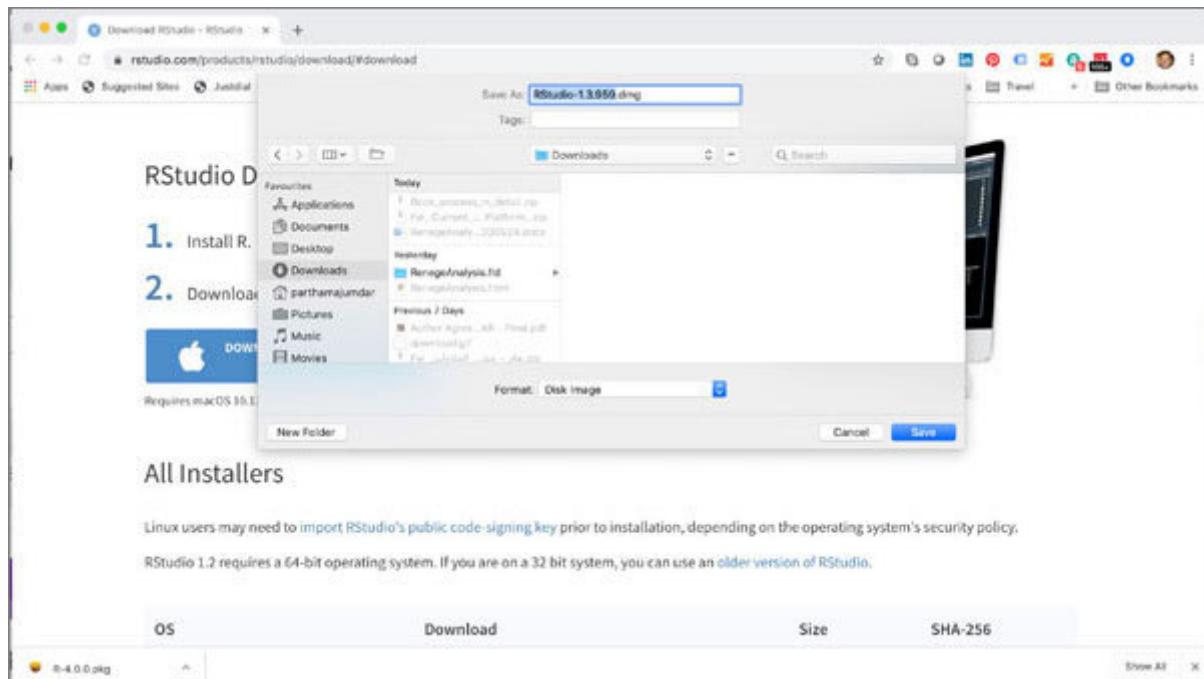
## Installing RStudio

On the web page shown in *figure* click on the download link under RStudio Desktop. You should get this web page as shown in the following screenshot:



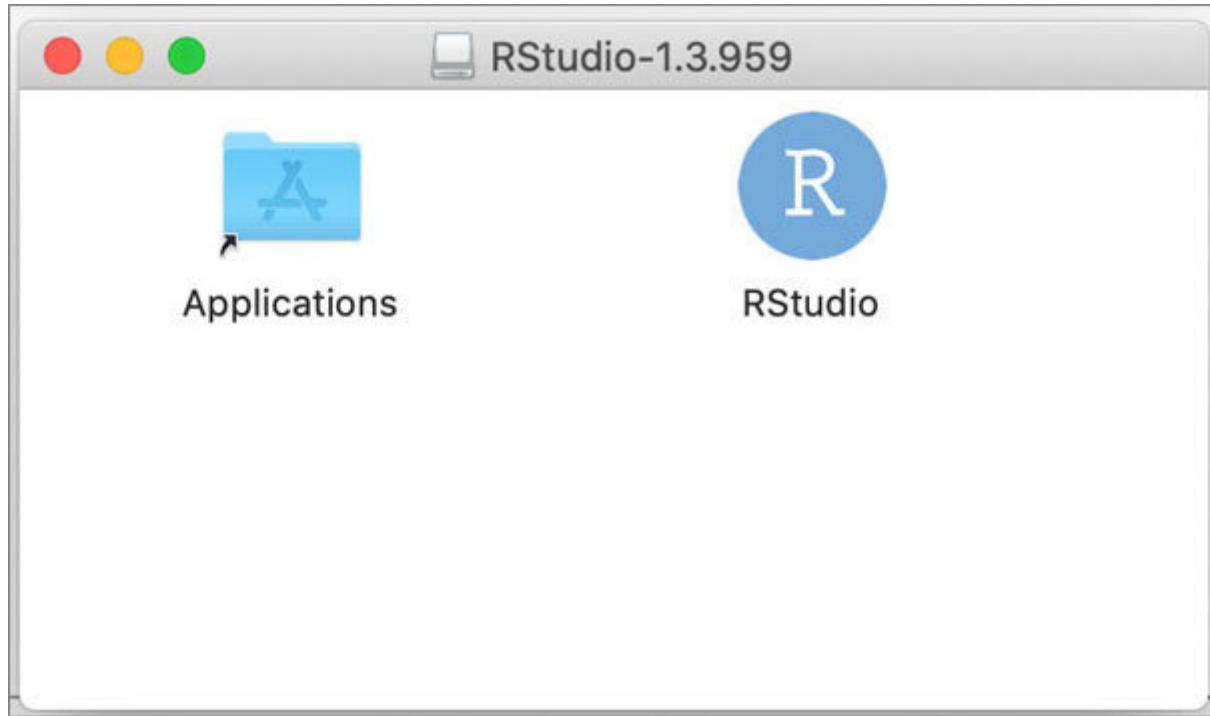
**Figure 1.20:** RStudio download link for Mac

Click on the download link as shown in *figure 1.20* to start the download of the software. Provide a location where the download should be saved (see in the following screenshot):



**Figure 1.21:** Save RStudio software to the disk

The download is a .dmg file (Disk Mirror Image). Once the software has been downloaded, locate the downloaded .dmg and run it. When the .dmg is run, the installation process starts. Once the installation is complete, the window shown in the following screenshot:



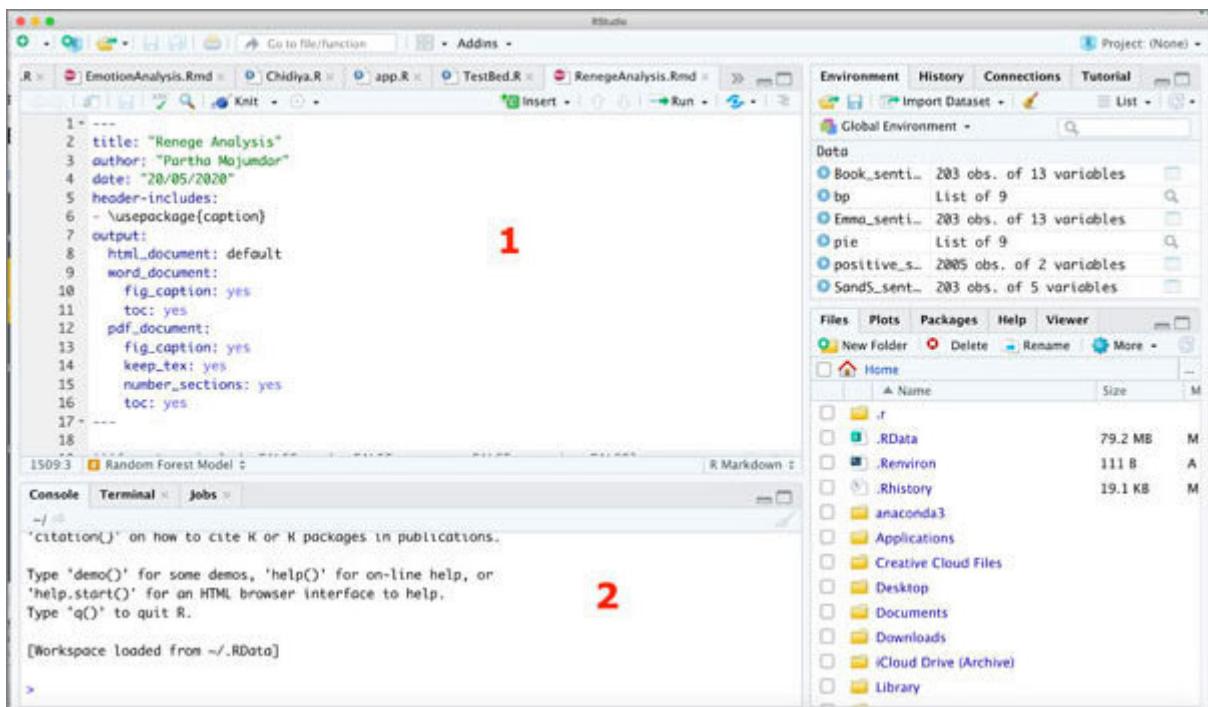
**Figure 1.22:** RStudio installation completed

In this window, drag the RStudio icon onto the applications icon. On doing so, the RStudio installation gets completed.

## Invoking RStudio

You can see the RStudio icon in [figure](#). Click on **Launchpad** and locate this icon. Then, click it to invoke RStudio.

The RStudio interface looks as shown in the following screenshot:



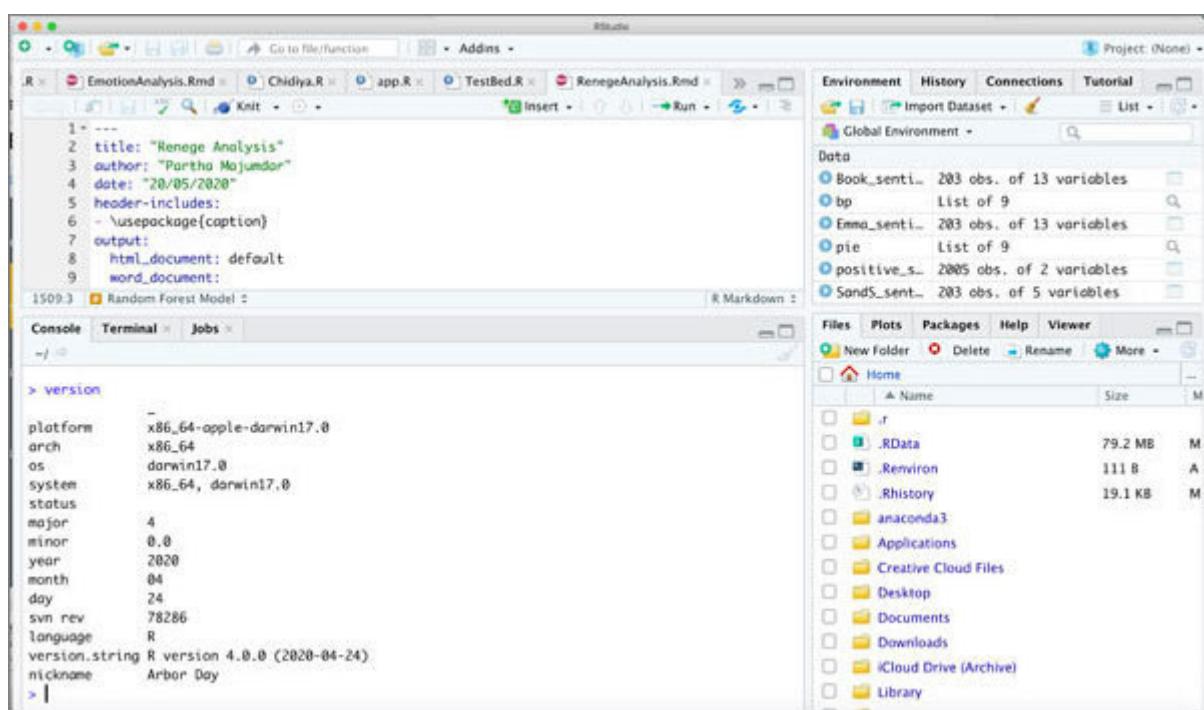
**Figure 1.23:** RStudio interface

As you can see, there are many parts in the RStudio interface. We will go through two aspects of the RStudio to get started. The area marked 1 is the area where the code is written. On the top-right corner of this panel, you can see a button titled On clicking

the **Run** button, the written program (or a part of the program) is executed.

The area marked 2 is the R Console. When the programs are run, the execution steps can be seen here. Also, it is possible to directly provide a command here and run it. Let us see an example for this.

To find out which version of R we are using, we can give the command `version`. In the picture below, you can see the command `version` in the R Console of the RStudio. The output of the command is shown after hitting



**Figure 1.24:** Running commands in R Console of RStudio

**The process to install RStudio Software on Windows and Linux machine is very similar. So, I will not repeat the process. You need to take care that the file types in Windows and Linux is different from that in Apple Mac. Also, the method of invoking an application in Windows and Linux is different from that in Apple Mac.**

## Introduction to packages

Packages are program collections which provide the libraries containing various functionalities. Using these libraries, it becomes possible to create various applications without bothering to program these essential features. The added advantage is that these libraries are thoroughly tested and tried by experts. Also, these libraries are routinely updated so that they function seamlessly with the new releases of R.

When R is installed, it comes with the default packages. Experts from various fields of studies regularly create and publish packages for specific requirements. These packages not only contain program units, but also data. For example, the package **tm** contains various program units (functions) essential for text mining. Also, for example, the package **janeaustenr** contains data regarding all the books of *Jane*

One of the most powerful features of R is that it has packages for almost all kinds of programming needs covering almost all domains. One of the most essential skills that R programmers need developing is to be able to find out the packages required for their purposes and know how to use those packages. Generally, elaborate documentation of each package can be found in the CRAN website. Also, there are various other websites which publish detailed documentation regarding all the published R packages.

If a package for an essential feature of the program being developed is not available in any of the published packages, they can be developed by the programmer. Once these program units have been developed, they can be compiled into a library. These libraries can be published to be included as a part of the R Software. We will cover how to create functions in this book. However, it is not the scope of this book to discuss the mechanism to create libraries and to publish such libraries to be included as part of the R software.

In general, all packages can be used without any need for licensing. However, there are certain packages which need licensing agreements if the developed software is to be used for commercial purposes.

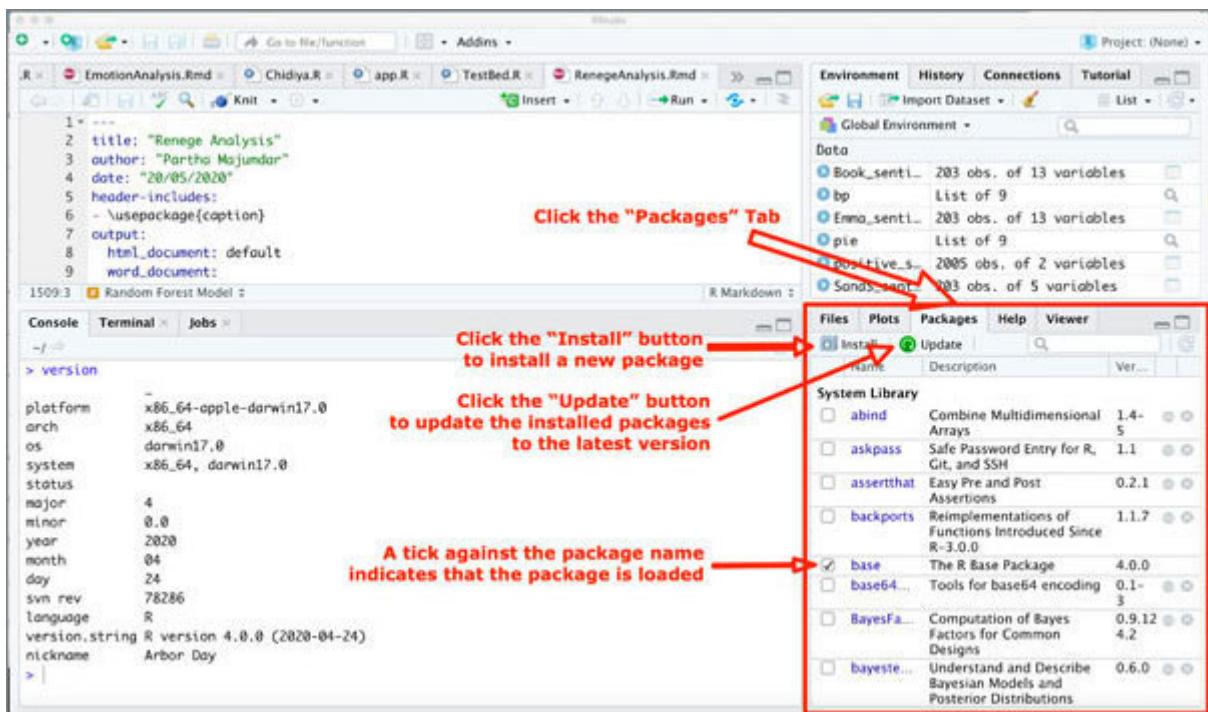
As a good practice, R programmers should cite the packages used by them in their programs. In the Appendix of this book, the mechanism to include citations is provided.

## Installing packages

We can install packages by using both the **Character User Interface** that is, using the **Command Line Interface** and the **Graphical User Interface**. We will discuss both the methods. We start our discussion using the GUI in RStudio.

## Installing packages using RStudio GUI

RStudio provides an easy interface for installing packages. In the screenshot below, the area of RStudio where all the installed packages are displayed is highlighted in the RED box. On clicking a package name, the documentation for the package is provided:



**Figure 1.25:** Package information in RStudio

On clicking the **Install** button as shown in [figure](#) a new package can be installed by providing the name of the package. To update the installed packages to the latest version, click the **Update** button.

To load a package (or library); tick the check box in front of the package.

## Installing packages using CLI

To install a package from the command line or programmatically, we need to use the **install.packages()** command. For example, to install the package the following command needs issuing:

```
install.packages("caret")
trying URL
'https://cran.rstudio.com/bin/macosx/contrib/4.0/caret_6.0-86.tgz'
Content type 'application/x-gzip' length 6246328 bytes (6.0 MB)
=====
downloaded 6.0 MB

The downloaded binary packages are in
/var/folders/bv/rqp385z157s87nvm3mg_tgnooooogn/T//RtmpJr4yKG/
downloaded_packages
```

We can check the packages installed currently using the command

```
row.names(installed.packages())
## [1] "abind"        "askpass"       "assertthat"
## [4] "backports"     "base"          "base64enc"
## [7] "BayesFactor"   "bayestestR"    "BH"
## [10] "bitops"        "boot"          "broom"
## [13] "callr"         "car"           "carData"
## [16] "caret"         "cellranger"   "CGPfunctions"
## ...
## [247] "utils"        "vctrs"         "viridis"
```

```
## [250] "viridisLite"  "whisker"    "withr"
## [253] "wordcloud"     "xfun"        "xml2"
## [256] "xtable"        "yaml"        "zip"
## [259] "zoo"
```

So, in a program, we can install a package, only if it is not already installed, by using a code as shown in the following code:

```
if!("caret" %in% installed.packages()) {
  install.packages("caret")
}
```

## [Loading libraries](#)

Only by installing the packages, the associated libraries are not available to the R environment and/or to the R programs. To make the libraries available, the libraries need to be loaded.

To load the libraries, we need the **library()** command. For example, to load the caret library, we need issuing the command as follows:

```
library(caret)
```

## Introduction to vector

Vector is a basic data structure in R. It contains elements of the same type. The data types of the elements of a vector can be logical, integer, double, character, complex, or raw.

We can create a vector of characters as follows:

```
c('h', 'e', 'l', 'l', 'o')
```

This vector contains 5 elements of the type character.

We can create a vector of strings as follows:

```
c('Welcome', 'to', 'R')
```

This vector contains 3 elements of the type character.

We can create a vector of integers as follows:

```
c(20, 32, 24, 32, 41, 34, 67, 96, 22, 52)
```

This vector contains 10 elements of the type integer.

## Assigning vectors to variables

We can store a vector in a variable. To store a vector in a variable, we need to use the <- (assignment) operator:

```
v_age <- c(20, 32, 24, 32, 41, 34, 67, 96, 22, 52)
```

The above command stores a vector containing 10 elements of the type integer in a variable named Let us consider that the elements of the vector, contains the ages of 10 persons.

Let us create another vector of salaries. Let us call this vector Suppose that the vector, contains the monthly salaries of people whose ages are stored in the vector

```
v_salary <- c(2000, 3200, 2400, 2000, 4000, 4200, 2000, 1000,  
5000, 10000)
```

## Checking the type of a vectors

A vector's type can be checked with the **typeof()** function:

```
typeof(v_salary)  
[1] "double"
```

## Checking the length of a vectors

A vector's length can be checked with the **length()** function:

```
length(v_salary)  
[1] 10
```

## Conducting statistical operations on a vectors

Various statistical operations can be conducted on a vector. Many of the functions for conducting statistical operations are available in the base R package. However, for more statistical functions, various libraries are essential.

We will see some statistical operations on a vector. We can find the **SUM** of all the elements of a vector using the **sum()** function:

```
sum(v_salary)  
[1] 35800
```

We can find the average of all the elements of a vector using the **mean()** function:

```
mean(v_salary)  
[1] 3580
```

We can find the standard deviation of all the elements of a vector using the **sd()** function:

```
sd(v_salary)  
[1] 2570.689
```

To see the summary of a vector, we can use the **summary()** function:

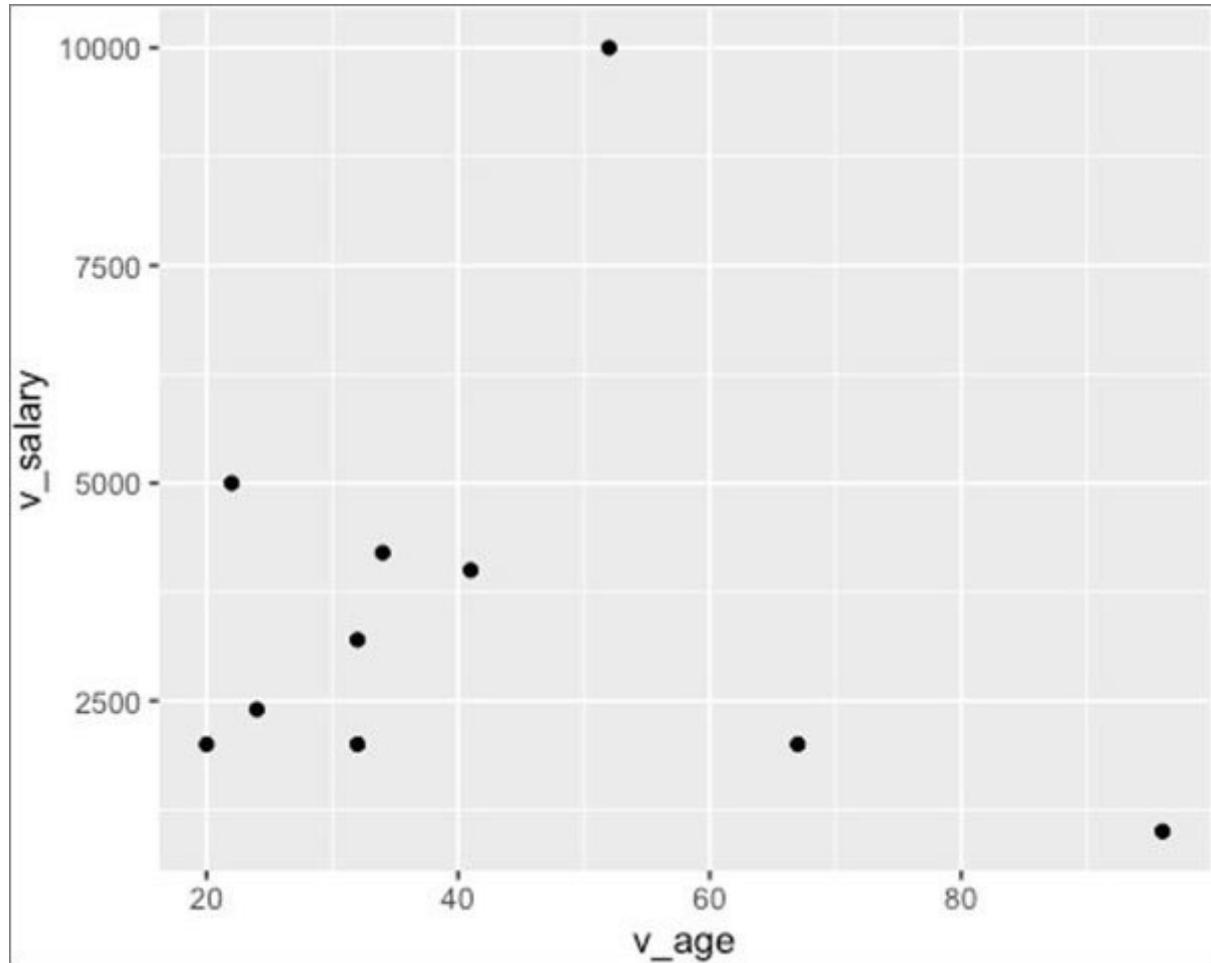
```
summary(v_age)
##   Min. 1st Qu. Median Mean 3rd Qu. Max.
## 20.00 26.00 33.00 42.00 49.25 96.00
```

To find the correlation between the elements of 2 vectors, we can use the **cor()** function. However, the **cor()** function is available in the **caret** package. So, first we need to load the **caret** library and then issue the **cor()** function:

```
library(caret)
cor(v_age, v_salary)
[1] -0.1320019
```

To create a scatter plot to visualize the relation between 2 vectors, we can use the **qplot()** function. The **qplot()** function is available in the **ggplot2** package:

```
library(ggplot2)
qplot(v_age, v_salary)
```



**Figure 1.26:** Scatter plot created using `qplot()`

## Conclusion

R is a very powerful and versatile language. Though it was initially created for statistical computing, it can now be used for many other types of computing. R is free software. Besides using R for developing applications, one can contribute libraries to R. As many people contribute libraries to R, R contains libraries for almost all types of computer programming for almost all domains.

In the next chapter, we will discuss the various operations that we can perform using R.

### Points to remember

R is free software. R comprises many packages. Most of the packages are available without the need for licensing. However, some packages may require licenses for commercial use.

R can be installed on any UNIX-like machine, Apple Mac, and Windows machines.

RStudio is an **Integrated Development Environment (IDE)** for R programming.

Packages in R can be installed using the **install.packages** command. To check for the installed packages, use the **installed.packages** command.

To load a library, use the **library** command.

A vector is a basic datatype in R. A vector contains elements of the same data type.

Use the **typeof()** function to find the type of a vector.

Use the **length()** function to find the length of a vector.

Use the **sum()** function to find the sum of all the elements of a vector containing numeric values.

Use the **mean()** function to find the average of all the elements of a vector containing numeric values.

Use the **sd()** function to find the standard deviation of all the elements of a vector containing numeric values.

Use the **summary()** function to find the summary of a vector containing numeric values.

Use the **cor()** function to find the correlation between 2 vectors.

Use the **qplot()** to plot a graph displaying the relation between 2 vectors.

### Multiple choice questions

What is the nickname of the R version used in this book?

Moby Dick

Arbor Day

Armistice Day

None of these

What is the RStudio version used in this book?

2.0.587

1.2.531

1.3.959

None of these

DMG stands for:

Disk Mirror imaGe

Disk Management Gate

Direct Messaging Game

None of these

The command to load libraries in R is:

load.libraries

load.packages

library

None of these

The command to list installed packages in R is:

list.packages

list.installed.packages

installed.packages

None of these

The function to find the sum of all the elements of a vector is:

SUM

sum

add

None of these

The **cor()** function is available in:

correlation package

caret package

statistics package

None of these

The function to find the average of all the elements of a vector is:

mean

average

avg

None of these

The function to find the standard deviation of all the elements of a vector is:

sd

stdev

std.dev

None of these

The **qplot()** command is available in the package:

graphics

plots

ggplot2

None of these

## Answers to MCQs

B

C

A

C

C

B

B

A

A

C

## Questions

What is type of file provided by RStudio for installing the software on a Windows machine?

Explain the process of installing the R Software on a Linux machine.

Find the documentation for the package **caret** on the CRAN website. List 5 functions available in the **caret** package.

## Key terms

**CLI:** Command Line Interface processes commands to a computer program in the form of lines of text. The program which handles the interface is called a command-line interpreter or command-line processor.

**CRAN:** The **Comprehensive R Archive Network (CRAN)** is a collection of sites which carry identical material consisting of the R distribution(s), the contributed extensions, the documentation for R, and binaries.

**CUI:** Character User Interface is a way for users to interact with computer programs. It works by allowing the user to issue commands as one or more lines of text to a program.

**DMG:** A file with the DMG file extension is an Apple Disk Mirror Image file, or sometimes called a Mac OS X Disk Mirror Image file, which is basically a digital reconstruction of a physical disk.

**GUI:** Graphical User Interface is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicators, etc. GUI is different from CUI which uses indicator such text-based user interfaces, typed command labels, or text navigation.

**IDE:** Integrated Development Environment provides an easy way of writing programs for a platform.

## CHAPTER 2

### Simple Operations Using R

In [chapter 1: Getting Started with R](#), we discussed how to install R and RStudio. Also, we discussed how to issue some basic command using R language.

In this chapter, we will discuss how to conduct various operations using R language. There is a lot which can be done using R language. We will not try to cover everything in an exhaustive manner. Instead, we will focus on the features of R language that are required for creating our programs for emotion analysis. The ground that we will cover should provide a good foundation to find out the features that we will not discuss.

## Structure

In this chapter, we will discuss the following topics:

Introduction to data frame

- Creating an empty data frame
- Creating a Data frame from vector(s)
- Renaming column(s) in a data frame
- Referencing row(s)/column(s) of a data frame
- Adding column(s) to a data frame
- Adding row(s) to a data frame
- Sorting a data frame
- Visualizing Data in a data frame

Reading Data from a comma separated value (CSV) File

Reading Data from a database



## Objectives

After studying this unit, you should be able to:

Create data frames and conduct operations on data frames

Read data from CSV Files

Read data from databases

Create graphs

## Introduction to data frame

Data frame is a data structure in R which has columns of data of different data types. A data frame has variables of a data set as columns and the observations as rows. A data frame is essential for storing data regarding any practical data set required for most analysis.

For example, to store the scores from an examination of a class of students, we may need to store the names and the roll numbers of the students along with the marks in each subject. While the names and the roll numbers would be of character data type, the marks in each subject would be of numeric data type (maybe integer data type).

Let us visit various operations on data frames in R including how to create data frames, how to view data frames, etc.

## Creating an empty data frame

An empty data frame can be created by initializing a data frame with a set of empty vectors using the `data.frame()` function. An example is shown in the following code:

```
v_marks <- data.frame(Name = character(),
Roll_Number = character(),
Company = character(),
Computer_Architecture = integer(),
Data_Analysis = integer(),
R_Programming = integer(),
stringsAsFactors = FALSE
)
```

In the above example, we created a data frame named `v_marks` containing 6 columns. The names of the columns are Name, Roll\_Number, Company, Computer\_Architecture, Data\_Analysis, and R\_Programming. The intention is to store the names of the students in the column Name and thus it is of character data type. We intend storing the roll numbers of the students in the column Roll\_Number and thus it is of character data type. We intend storing the name of the company of the students in the column Company and thus it is of character data type. We intend storing the marks in the subjects computer architecture, data analysis, computer architecture, data analysis and r programming in the columns Computer\_Architecture, Data\_Analysis, and R\_Programming respectively and thus they are of integer data type.

**The basic data types in R are character, numeric, integer, complex and logical.**

To view a data frame, we need to provide the name of the data frame as shown in the following code:

### v\_marks

```
## [1] Name          Roll_Number      Company  
## [4] Computer_Architecture Data_Analysis  
R_Programming  
## <0 rows> (or 0-length row.names)
```

Every data frame has a structure. To view the structure of a data frame, we need to use the `str()` function as shown in the following code:

### str(v\_marks)

```
## 'data.frame':    0 obs. of  6 variables:  
## $ Name          : chr  
## $ Roll_Number   : chr  
## $ Company       : chr  
## $ Computer_Architecture: int  
## $ Data_Analysis: int  
## $ R_Programming: int
```

The `str()` function displays the detailed structure of the data frame. It shows that the data frame `v_marks` has 0 observations. Also, it

shows that the data frame v\_marks has 6 columns (variables). The data type of each column (or variable) is specified.

In general, we can see the type of a variable in R using the class() function. So, we can check the type of the variable v\_marks as shown in the following code:

```
class(v_marks)
## [1] "data.frame"
```

An empty data frame, without any columns, can be created as follows using the data.frame() function:

```
v_empty_data_frame <- data.frame()
str(v_empty_data_frame)
## 'data.frame':    0 obs. of  0 variables
```

In the latest version of R (the version we are using in this book is 4.0.0), the recommended method to create an empty data frame (or any data frame) is by using the **tibble()** function. **tibble()** function is available in the library tibble. An example is shown in the following code:

```
library(tibble)
v_tibble_marks <- tibble(Name = character(),
Roll_Number = character(),
Company = character(),
Computer_Architecture = integer(),
Data_Analysis = integer(),
```

There are several advantages of creating a data frame using the `tibble()` function. One of the major advantages that we can discuss at this stage is that the `tibble()` function does not perform any automatic data type conversion while creating a data frame.

The documentation for the `tibble` library can be found at the URL

## Creating a Data Frame from vector(s)

In [Chapter 1: Getting Started with](#) we discussed how to create a vector. When we have a set of vectors, we can create a data frame from these vectors.

Let us consider that we have 6 vectors as follows:

```
v_names <- c("Amarendra", "Anita", "Anuj", "Arindam",
"Deepshree", "Kousik")
v_roll_number <- c('101', '102', '103', '104', '105', '106')
v_company <- c('Atos', 'MC', 'Atos', 'MC', 'IBM', 'Atos')
v_marks_ca <- c(87, 86, 92, 85, 77, 90)
v_marks_da <- c(95, 86, 90, 85, 80, 82)
v_marks_rp <- c(60, 65, 72, 55, 54, 80)
```

We can create a data frame using these vectors as follows:

```
v_marks <- data.frame(Name = v_names,
Roll_Number = v_roll_number,
Company = v_company,
Computer_Architecture = v_marks_ca,
Data_Analysis = v_marks_da,
R_Programming = v_marks_rp,
stringsAsFactors = FALSE
)
str(v_marks)
```

```

## 'data.frame':   6 obs. of  6 variables:
## $ Name           : chr  "Amarendra" "Anita" "Anuj"
## "Arindam" ...
## $ Roll_Number    : chr  "101" "102" "103" "104" ...
## 
## $ Company        : chr  "Atos" "MC" "Atos" "MC"
...
## $ Computer_Architecture: num  87 86 92 85 77 90
## $ Data_Analysis   : num  95 86 90 85 80 82
## $ R_Programming   : num  60 65 72 55 54 80

```

We can view the data frame and get the following result:

v\_marks

	Name	Roll_Number	Company	Computer_Architecture
## 1	Amarendra	101	Atos	87
		95		
## 2	Anita	102	MC	86
		86		
## 3	Anuj	103	Atos	92
		90		
## 4	Arindam	104	MC	85
		85		
## 5	Deepshree	105	IBM	77
		80		

```

## 6
Kousik      106          Atos          90
82
## R_Programming
## 1          60
## 2          65
## 3          72
## 4          55
## 5          54

## 6          80

```

In the previous example, we had given the parameter stringsAsFactors as equal to Now let us see what happens when we give this parameter as

```

v_marks <- data.frame(Name = v_names,
Roll_Number = v_roll_number,
Company = v_company,
Computer_Architecture = v_marks_ca,
Data_Analysis = v_marks_da,
R_Programming = v_marks_rp,
stringsAsFactors = TRUE
)
str(v_marks)
## 'data.frame'           : 6 obs. of  6 variables:
## $ Name                 : Factor w/ 6 levels
"Amarendra","Anita",.. : 1 2 3 4 5 6
## $ Roll_Number          : Factor w/ 6 levels
"101","102","103",.. : 1 2 3 4 5 6

```

```

## $ Company : Factor w/ 3 levels
"Atos","IBM","MC" : 1 3 1 3 2 1
## $ Computer_Architecture : num 87 86 92 85 77 90
## $ Data_Analysis : num 95 86 90 85 80 82
## $ R_Programming : num 60 65 72 55 54 80

```

You notice that all the columns with character type data are now factors. For a factor, R takes all the unique values of the column and creates a level for each unique value. You notice this best for the column Each unique value has been given a level number.

R conducts this data type conversion when we create data frames using the `data.frame()` function unless we explicitly provide the parameter `stringsAsFactors =`

Further, notice that each data point in the column is indicated by the level

Viewing the data frame, we notice that there is no difference from what we saw before. However, we will find out later that there are differences in how this data frame can be used:

### v\_marks

```

##   Name      Roll_Number    Company    Computer_Archit
ecture    Data_Analysis
## 1
Amarendra      101        Atos       87

```

```

## 2
Anita          102      MC           86
                    86

## 3
Anuj           103      Atos         92
                    90

## 4
Arindam        104      MC           85
                    85

## 5
Deepshree      105      IBM          77
                    80

## 6
Kousik         106      Atos         90
                    82

## R_Programming
## 1            60
## 2            65
## 3            72

## 4            55
## 5            54
## 6            80

```

We can create the data frame from vectors using the `tibble()` function as well:

```

library(tibble)
v_tibble_marks <- tibble(Name = v_names,
Roll_Number = v_roll_number,
Company = v_company,

```

We get the following result when we view the data frame:

```
v_tibble_marks  
## # A tibble: 6 x 6  
  
##      Name Roll_Number Company Computer_Architecture  
Data_Analysis R_Programming  
##  
## 1 Amarendra 101 Atos 87 95 60  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72  
## 4 Arindam 104 MC 85 85 55  
## 5 Deepshree 105 IBM 77 80 54  
## 6 Kousik 106 Atos 90 82 80
```

## [Renaming column\(s\) in a data frame](#)

We can rename the columns of a data frame. There are many ways to do this. I recommend renaming columns in a data frame using the `rename()` function available in the `dplyr` library.

Let us do the following:

Rename the `Roll_Number` column to

Rename the `Computer_Architecture` column to

Rename the `Data_Analysis` column to

Rename the `R_Programming` column to

The syntax for the `rename()` function is as follows:

```
rename(data_frame, new_column_name = old_column_name)
```

Let us rename the `Roll_Number` column:

```
library(dplyr)  
v_marks <- rename(v_marks, "RN" = "Roll_Number")
```

Here, we have renamed one column in the data frame and assigned the result to the same variable. Let us view the data frame:

```
v_marks  
## Name RN Company Computer_Architecture Data_Analysis  
R_Programming  
## 1 Amarendra 101 Atos 87 95 60  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72  
## 4 Arindam 104 MC 85 85 55  
## 5 Deepshree 105 IBM 77 80 54  
## 6 Kousik 106 Atos 90 82 80
```

We can rename multiple columns in a data frame in one command by providing the renaming details as a vector. An example is shown in the following code:

```
library(dplyr)  
v_marks <- rename(v_marks, c("CA" = "Computer_Architecture",  
"DA" = "Data_Analysis", "RP" = "R_Programming"))
```

The data frame now looks as follows:

```
v_marks  
## Name RN Company CA DA RP  
## 1 Amarendra 101 Atos 87 95 60  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72
```

```
## 4 Arindam 104 MC 85 85 55  
## 5 Deepshree 105 IBM 77 80 54  
## 6 Kousik 106 Atos 90 82 80
```

The documentation for the `dplyr` library can be found at the URL

## Referencing row(s)/column(s) of a data frame

There are several ways the data in a data frame can be referenced. We will discuss two methods of referencing data in a data frame. The choice of the referencing method depends on the programming requirements. While some referencing methods allow for static referencing, others allow for dynamic referencing. By dynamic referencing, it is implied that the data can be accessed by passing the parameters programmatically.

We can reference data in a data frame by using the square brackets ([row, column]). Inside the square brackets, we can provide the specific row number(s) and the specific column number(s) of the data we need referencing. For example, to see the data in row number 1 and column number 1 of the data frame we can write the program as follows:

```
v_marks[1, 1]  
## [1] Amarendra  
## Levels: Amarendra Anita Anuj Arindam Deepshree Kousik
```

We will revisit this a little later in the book.

For viewing all the columns of a row, we can omit the column parameter. For example, to see all the columns of the 1st records (1st row), we can issue the command as follows:

```
v_marks[1,]  
## Name N Company CA DA RP  
## 1 Amarendra 101 Atos 87 95 60
```

For viewing a set of columns of a row, we can provide the column numbers as a vector. For example, to see the name, roll number, and company for Amarendra (record number 1 or row number 1), we can issue the command as follows:

```
v_marks[1, c(1, 2, 3)]  
## Name RN Company  
## 1 Amarendra 101 Atos
```

We can rearrange the columns as shown below:

```
v_marks[1, c(2, 1, 3)]  
## RN Name Company  
## 1 101 Amarendra Atos
```

If the column numbers are continuous, we can use colon (:) as shown in the following code:

```
v_marks[1, 1:3]  
## Name RN Company  
## 1 Amarendra 101 Atos
```

We can combine the methods of using a vector and a colon as shown in the following code:

```
v_marks[1, c(1, 4:6)]  
## Name CA DA RP  
## 1 Amarendra 87 95 60
```

Lastly, we can omit the row number row number parameter to see all the rows as shown in the following code:

```
v_marks[, c(1, 4:6)]  
## Name CA DA RP  
## 1 Amarendra 87 95 60  
## 2 Anita 86 86 65  
## 3 Anuj 92 90 72  
  
## 4 Arindam 85 85 55  
## 5 Deepshree 77 80 54  
## 6 Kousik 90 82 80
```

All the referencing methods that we applied for referencing columns apply to referencing rows as well. For referencing a set of rows, we can supply the row numbers row numbers as a vector as shown in the following code:

```
v_marks[c(1, 6, 3),]  
## Name RN Company CA DA RP  
## 1 Amarendra 101 Atos 87 95 60  
## 6 Kousik 106 Atos 90 82 80  
## 3 Anuj 103 Atos 92 90 72
```

If the rows are continuous, then we can use colon (:) as shown below:

```
v_marks[c(1:4),]  
## Name RN Company CA DA RP  
## 1 Amarendra 101 Atos 87 95 60  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72  
## 4 Arindam 104 MC 85 85 55
```

We can combine using a vector and colon (:) as shown in the following code:

```
v_marks[c(5, 1:3),]  
## Name RN Company CA DA RP  
## 5 Deepshree 105 IBM 77 80 54  
## 1 Amarendra 101 Atos 87 95 60  
  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72
```

Lastly, we can apply all the referencing discussed so far on both rows and columns together as shown in the following code:

```
v_marks[c(1:3, 6), c(1, 4:6)]  
## Name CA DA RP  
## 1 Amarendra 87 95 60  
## 2 Anita 86 86 65  
## 3 Anuj 92 90 72  
## 6 Kousik 90 82 80
```

We can reference a column using the dollar operator. For example, to get the list of marks of computer architecture, we can issue the following command:

```
v_marks$CA  
## [1] 87 86 92 85 77 90
```

Notice that the output returned is a vector. We can verify this using the `class()` function:

```
class(v_marks$CA)  
## [1] "numeric"
```

We can fetch data for a column for a set of rows using colon (`:`) as follows:

```
v_marks[1:3]$Company  
## [1] Atos MC Atos  
## Levels: Atos IBM MC
```

We can fetch data for a column for a set of rows combining a vector and a colon (`:`) as follows:

```
v_marks[c(1, 3:5)]$Company  
## [1] Atos Atos MC IBM  
## Levels: Atos IBM MC
```

## Referencing a data frame based on conditions

We can reference the rows of a data frame based on a condition. For this, we will discuss two functions - which() and Before we discuss the which() and subset() functions, we need to know more about the logical operators required for forming conditions.

R provides **Double Equal To** (==) operator for equating two values. R provides >, >=, <, <=, and != operators for greater than, greater than & equal to, less than, less than & equal to and not equal to operations, respectively.

R provides the Pipe () operator for performing the element-wise OR operation, the Ampersand (&) operator for performing the element-wise AND operation, and the Exclamation (!) operator for performing the NOT operations.

**Single Pipe () operator is different from Double Pipe (||) operator in R. Similarly, Single Ampersand (&) operator is different from Double Ampersand (&&) operator in R.**

Let us start the discussion by fetching the record for roll number 105 using the which() function. The command is as follows:

```
v_marks[which(v_marks$RN == "105"),]  
## Name RN Company CA DA RP  
## 5 Deepshree 105 IBM 77 80 54
```

Notice that for referencing column RN, we need to specify the data frame. We can avoid this if we attach the data frame using the attach() function as shown in the following code:

```
attach(v_marks)
v_marks[which(RN == "105"),]

## Name RN Company CA DA RP
## 5 Deepshree 105 IBM 77 80 54
```

For fetching the record(s) of the students from the company Atos, we can issue the command as follows:

```
v_marks[which(Company == "Atos"),]
## Name RN Company CA DA RP
## 1 Amarendra 101 Atos 87 95 60
## 3 Anuj 103 Atos 92 90 72
## 6 Kousik 106 Atos 90 82 80
```

For fetching the record(s) of the students who have scored more than 70 in R programming, we can issue the command as follows:

```
v_marks[which(RP > 70),]
## Name RN Company CA DA RP
## 3 Anuj 103 Atos 92 90 72
## 6 Kousik 106 Atos 90 82 80
```

**Once a data frame is attached using the attach() function, it remains attached till the data frame is detached using the detach() function.**

For fetching the record(s) of the students who have scored more than 60 in R programming and more than 80 in computer architecture, we can issue the command as follows:

```
v_marks[which(RP > 60 & CA > 80),]  
## Name RN Company CA DA RP  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72  
  
## 6 Kousik 106 Atos 90 82 80
```

For fetching the record(s) of the students who either work in IBM or who have scored more than 60 in R programming and more than 80 in computer architecture, we can issue the command as follows:

```
v_marks[which((Company == "IBM") | (RP > 60 & CA > 80)),]  
## Name RN Company CA DA RP  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72  
## 5 Deepshree 105 IBM 77 80 54  
## 6 Kousik 106 Atos 90 82 80
```

As you would have realized by now, the output of all the above commands is a data frame. So, if we assign the output from

these commands to a variable, the resulting variable will also be a data frame. Let me illustrate this with an example:

```
v_subset <- v_marks[which(RP > 70 & CA > 80),]  
v_subset  
## Name RN Company CA DA RP  
## 3 Anuj 103 Atos 92 90 72  
## 6 Kousik 106 Atos 90 82 80  
class(v_subset)  
## [1] "data.frame"
```

We can achieve the above results with the `subset()` function as well. Let us go through the illustration. For fetching the record(s) of the students from the company Atos, we can issue the command as follows:

```
subset(v_marks, (Company ==  
## Name RN Company CA DA RP  
## 1 Amarendra 101 Atos 87 95 60  
## 3 Anuj 103 Atos 92 90 72  
## 6 Kousik 106 Atos 90 82 80
```

For fetching the record(s) of the students who have scored more than 60 in R programming and more than 80 in computer architecture, we can issue the command as follows:

```
subset(v_marks, (RP > 60 & CA > 80))  
## Name RN Company CA DA RP  
## 2 Anita 102 MC 86 86 65
```

```
## 3 Anuj 103 Atos 92 90 72  
## 6 Kousik 106 Atos 90 82 80
```

For fetching the record(s) of the students who either work in IBM or who have scored more than 60 in R programming, and more than 80 in computer architecture, we can issue the command as follows:

```
v_subset <- subset(v_marks, ((Company == "IBM") | (RP > 60 &  
CA > 80)))  
v_subset  
## Name RN Company CA DA RP  
## 2 Anita 102 MC 86 86 65  
## 3 Anuj 103 Atos 92 90 72  
## 5 Deepshree 105 IBM 77 80 54  
## 6 Kousik 106 Atos 90 82 80  
class(v_subset)  
## [1] "data.frame"
```

## [Adding column\(s\) to a data frame](#)

We can add column(s) to a data frame in many ways. The different ways are required for different requirements. We will discuss some of the techniques required for our purpose.

First, let us add a column for Marks in Shiny Programming and name this column as SP. We will consider that we have the data for this column in a vector.

For performing this action, we can use the cbind() function as shown in the following code:

```
v_marks_sp <- c(55, 67, 71, 54, 80, 69)
v_marks <- cbind(v_marks, SP = v_marks_sp)
```

The data frame v\_marks looks as follows:

### **v\_marks**

```
## Name RN Company CA DA RP SP
## 1 Amarendra 101 Atos 87 95 60 55
## 2 Anita 102 MC 86 86 65 67
## 3 Anuj 103 Atos 92 90 72 71
## 4 Arindam 104 MC 85 85 55 54
## 5 Deepshree 105 IBM 77 80 54 80
## 6 Kousik 106 Atos 90 82 80 69
```

Now, let us create a calculated column. We can create calculated columns using the `cbind()` function. However, we will explore another technique. We will add a column containing the total marks scored across all the subjects. Let us name this column as Total. See the syntax below to create this column:

```
v_marks$Total <- v_marks$CA + v_marks$DA + v_marks$RP +  
v_marks$SP
```

The data frame v\_marks looks as follows:

```
v_marks  
## Name RN Company CA DA RP SP Total  
## 1 Amarendra 101 Atos 87 95 60 55 297  
## 2 Anita 102 MC 86 86 65 67 304  
## 3 Anuj 103 Atos 92 90 72 71 325  
## 4 Arindam 104 MC 85 85 55 54 279  
## 5 Deepshree 105 IBM 77 80 54 80 291  
## 6 Kousik 106 Atos 90 82 80 69 321
```

Now let us add a calculated column using the cbind() function. Let us add a column to store the percentage marks and name the column as

```
v_marks <- cbind(v_marks, Percentage = (v_marks$Total/400*100))
```

The data frame `y_marks` looks as follows:

```
v_marks

## Name RN Company CA DA RP SP Total Percentage
## 1 Amarendra 101 Atos 87 95 60 55 297 74.25
## 2 Anita 102 MC 86 86 65 67 304 76.00
## 3 Anuj 103 Atos 92 90 72 71 325 81.25
## 4 Arindam 104 MC 85 85 55 54 279 69.75
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75
## 6 Kousik 106 Atos 90 82 80 69 321 80.25
```

We can add columns whose values are computed based on a condition or conditions. For example, let us say that we want to add a column called Grade such that if the percentage marks is greater than 80, then the grade should be A if the percentage marks is greater than 70 and less than or equal to 80, then the grade should be B if the percentage marks is greater than 60 and less than or equal to 70, then the grade should be C otherwise, the grade should be D.

We can use either of the two techniques discussed above. I will demonstrate one technique. You can try the other technique yourself:

```
v_marks$Grade <- ifelse(v_marks$Percentage > 80, "A",
ifelse(v_marks$Percentage > 70, "B",
ifelse(v_marks$Percentage > 60, "C", "D"))
)
)
v_marks

## Name RN Company CA DA RP SP Total Percentage Grade
```

## 1 Amarendra 101 Atos 87 95 60 55 297 74.25 B  
## 2 Anita 102 MC 86 86 65 67 304 76.00 B  
## 3 Anuj 103 Atos 92 90 72 71 325 81.25 A  
## 4 Arindam 104 MC 85 85 55 54 279 69.75 C  
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75 B  
## 6 Kousik 106 Atos 90 82 80 69 321 80.25 A

## [Adding row\(s\) to a data frame](#)

We can add row(s) to a data frame using the rbind() function.

The rbind() function can used in different ways. However, I will discuss only one technique.

To add new records, we can create a new data frame containing the new records and then add the new records to the existing data frame using the rbind() function. So, we first create a new data frame named v\_new\_students containing the data for the new students as shown in the following code:

```
v_new_student_names <- c("Ranoo", "Babita", "Ujwala")
v_new_student_rn <- c("108", "109", "110")
v_new_student_company <- c("MC", "MC", "Gokuldas")
v_new_student_ca <- c(50, 45, 55)
v_new_student_da <- c(65, 47, 72)
v_new_student_rp <- c(47, 42, 43)
v_new_student_sp <- c(42, 40, 45)
v_new_student_total <- v_new_student_ca + v_new_student_da +
v_new_student_rp + v_new_student_sp
v_new_student_percent <- v_new_student_total / 400 * 100
v_new_student_grade <- ifelse(v_new_student_percent > 80, "A",
ifelse(v_new_student_percent > 70, "B",
ifelse(v_new_student_percent > 60, "C", "D"))
)
)
v_new_students <- data.frame(Name = v_new_student_names,
```

```

RN = v_new_student_rn,
Company = v_new_student_company,
CA = v_new_student_ca,

DA = v_new_student_da,
RP = v_new_student_rp,
SP = v_new_student_sp,
Total = v_new_student_total,
Percentage = v_new_student_percent,
Grade = v_new_student_grade
)
v_new_students
## Name RN Company CA DA RP SP Total Percentage Grade
## 1 Ranoo 108 MC 50 65 47 42 204 51.00 D
## 2 Babita 109 MC 45 47 42 40 174 43.50 D
## 3 Ujwala 110 Gokuldas 55 72 43 45 215 53.75 D

```

Before using the rbind() function to add one or more records, we must ensure that the data frame does not contain any factors. This is because the presence of factor(s) will disallow any value in the factor columns which are not already present in one of the levels.

All our factor columns are columns containing strings. For converting these factor columns to character columns, we can use the as.character() function as shown in the following code:

```

v_marks$Name <- as.character(v_marks$Name)
v_marks$RN <- as.character(v_marks$RN)
v_marks$Company <- as.character(v_marks$Company)

```

Check the structure of Now, we add the new records (or rows) to v\_marks as shown in the following code. You should have realized that the columns in both the data frames should be identical:

```
v_marks <- rbind(v_marks, v_new_students, stringsAsFactors =  
FALSE)  
v_marks  
## Name RN Company CA DA RP SP Total Percentage Grade  
## 1 Amarendra 101 Atos 87 95 60 55 297 74.25 B  
## 2 Anita 102 MC 86 86 65 67 304 76.00 B  
## 3 Anuj 103 Atos 92 90 72 71 325 81.25 A  
## 4 Arindam 104 MC 85 85 55 54 279 69.75 C  
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75 B  
## 6 Kousik 106 Atos 90 82 80 69 321 80.25 A  
## 7 Ranoo 108 MC 50 65 47 42 204 51.00 D  
## 8 Babita 109 MC 45 47 42 40 174 43.50 D  
## 9 Ujwala 110 Gokuldas 55 72 43 45 215 53.75 D
```

Now, let us convert the column company to a factor. To convert a column to a factor, we need to use the as.factor() function:

```
v_marks$Company <- as.factor(v_marks$Company)  
str(v_marks)  
## 'data.frame': 9 obs. of 10 variables:  
## $ Name : chr "Amarendra" "Anita" "Anuj" "Arindam"  
...  
## $ RN : chr "101" "102" "103" "104" ...  
## $ Company : Factor w/ 4 levels "Atos","Gokuldas",..: 1 4 1  
4 3 1 4 4 2  
## $ CA : num 87 86 92 85 77 90 50 45 55
```

```
## $ DA      : num  95 86 90 85 80 82 65 47 72
## $ RP      : num  60 65 72 55 54 80 47 42 43
## $ SP      : num  55 67 71 54 80 69 42 40 45
## $ Total    : num  297 304 325 279 291 321 204 174 215
## $ Percentage: num  74.2 76 81.2 69.8 72.8 ...

## $ Grade    : chr  "B" "B" "A" "C" ...
```

## Sorting a data frame

We can sort a data frame using the `arrange()` function available in the `dplyr` package. Let's go through a few examples to understand the concepts. The basic syntax of the `arrange()` function is as shown below:

```
arrange(data_frame, column1, column2, ...)
```

etc. are the columns by which to sort the data frame. First, let us sort the data frame by total marks. We need to give the following command:

```
library(dplyr)
arrange(v_marks, Total)
## # Name RN Company CA DA RP SP Total Percentage Grade
## 1 Babita 109 MC 45 47 42 40 174 43.50 D
## 2 Ranoo 108 MC 50 65 47 42 204 51.00 D
## 3 Ujwala 110 Gokuldas 55 72 43 45 215 53.75 D
## 4 Arindam 104 MC 85 85 55 54 279 69.75 C
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75 B
## 6 Amarendra 101 Atos 87 95 60 55 297 74.25 B
## 7 Anita 102 MC 86 86 65 67 304 76.00 B
## 8 Kousik 106 Atos 90 82 80 69 321 80.25 A
## 9 Anuj 103 Atos 92 90 72 71 325 81.25 A
```

Notice that by default, the `arrange()` function sorts in the ascending order. To sort in the descending order, we need to use the `desc()` function as shown in the following code:

```
arrange(v_marks, desc(Total))
## Name RN Company CA DA RP SP Total Percentage Grade

## 1 Anuj 103 Atos 92 90 72 71 325 81.25 A
## 2 Kousik 106 Atos 90 82 80 69 321 80.25 A
## 3 Anita 102 MC 86 86 65 67 304 76.00 B
## 4 Amarendra 101 Atos 87 95 60 55 297 74.25 B
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75 B
## 6 Arindam 104 MC 85 85 55 54 279 69.75 C
## 7 Ujwala 110 Gokuldas 55 72 43 45 215 53.75 D
## 8 Ranoo 108 MC 50 65 47 42 204 51.00 D
## 9 Babita 109 MC 45 47 42 40 174 43.50 D
```

We can sort by more than 1 column. For example, to sort the data frame by company and within the company by the name of the student, we can issue the command as follows:

```
arrange(v_marks, Company, Name)
## Name RN Company CA DA RP SP Total Percentage Grade

## 1 Amarendra 101 Atos 87 95 60 55 297 74.25 B
## 2 Anuj 103 Atos 92 90 72 71 325 81.25 A
## 3 Kousik 106 Atos 90 82 80 69 321 80.25 A
## 4 Ujwala 110 Gokuldas 55 72 43 45 215 53.75 D
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75 B
## 6 Anita 102 MC 86 86 65 67 304 76.00 B
## 7 Arindam 104 MC 85 85 55 54 279 69.75 C
## 8 Babita 109 MC 45 47 42 40 174 43.50 D
```

## 9 Ranoo 108 MC 50 65 47 42 204 51.00 D

You realize by now that the `arrange()` function returns a data frame. So, we can store the result in a variable and use it later, if required:

```
v_by_grade <- arrange(v_marks, Grade, desc(Total))
```

## v\_by\_grade

```

## Name RN Company CA DA RP SP Total Percentage Grade
## 1 Anuj 103 Atos 92 90 72 71 325 81.25 A
## 2 Kousik 106 Atos 90 82 80 69 321 80.25 A
## 3 Anita 102 MC 86 86 65 67 304 76.00 B
## 4 Amarendra 101 Atos 87 95 60 55 297 74.25 B
## 5 Deepshree 105 IBM 77 80 54 80 291 72.75 B
## 6 Arindam 104 MC 85 85 55 54 279 69.75 C
## 7 Ujwala 110 Gokuldas 55 72 43 45 215 53.75 D
## 8 Ranoo 108 MC 50 65 47 42 204 51.00 D
## 9 Babita 109 MC 45 47 42 40 174 43.50 D

```

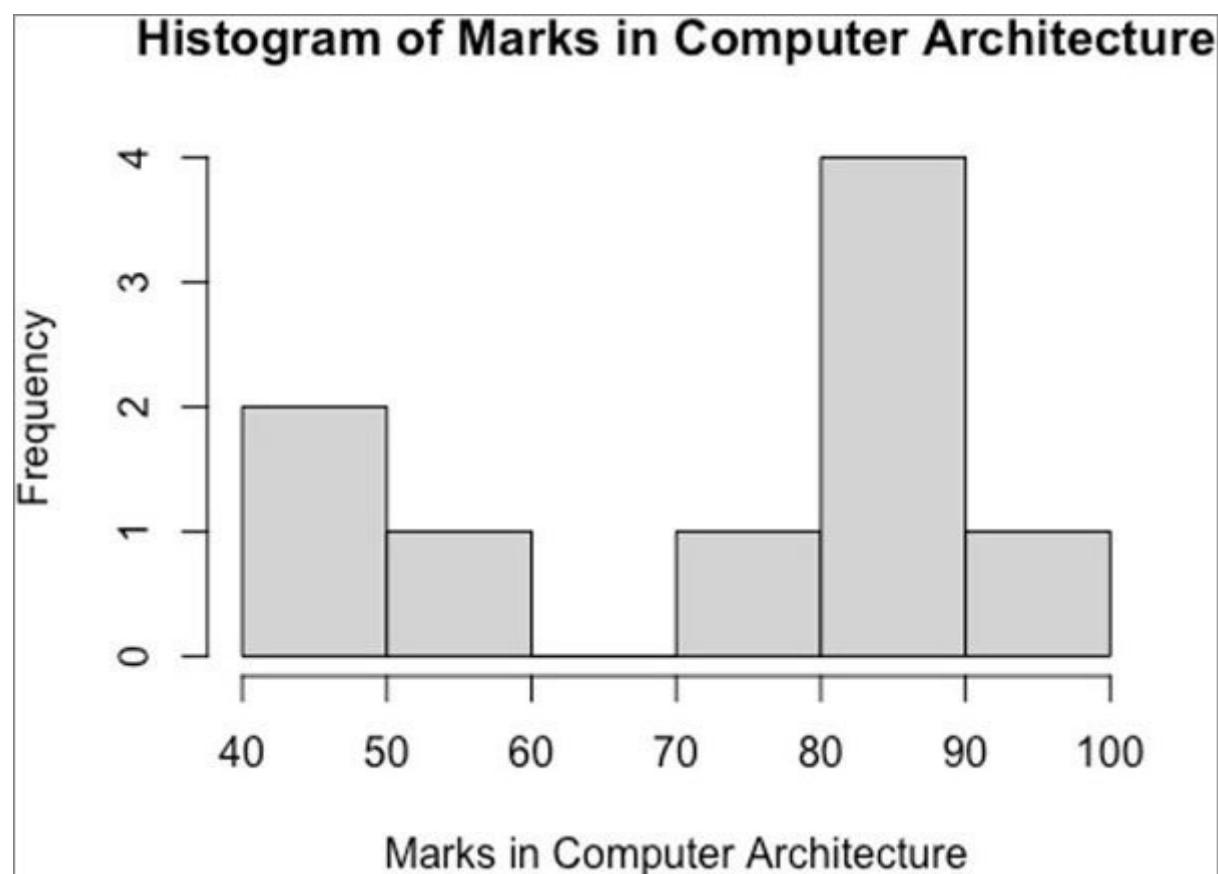
## Visualizing data in a data frame

Now, let us create some visualizations for the data we have in our data frame.

## Histogram

Histograms provide a visualization of the distribution. We start with creating a histogram for the marks obtained in computer architecture. We can create the histogram using the `hist()` function as shown in the following code:

```
hist(v_marks$CA,  
main = "Histogram of Marks in Computer Architecture",  
xlab = "Marks in Computer Architecture"  
)
```

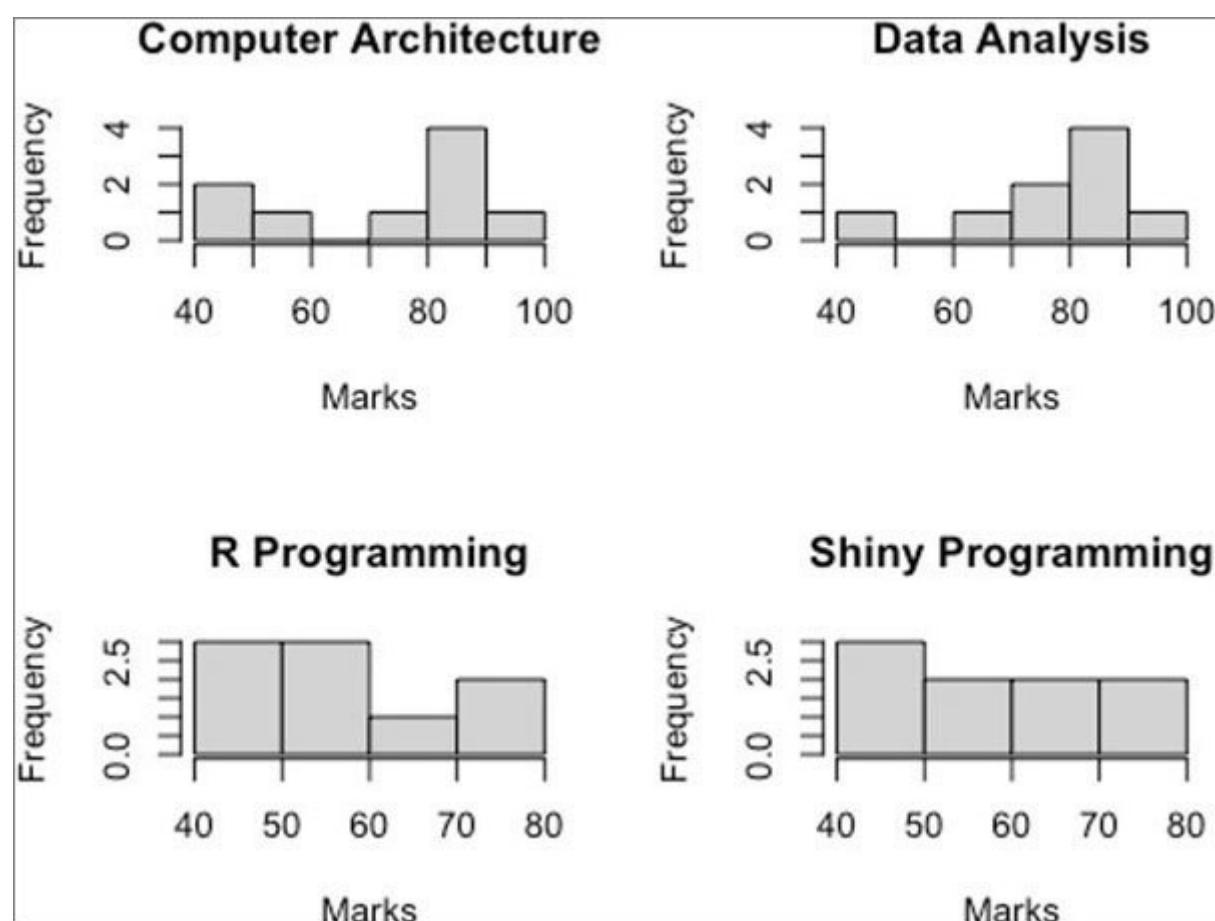


**Figure 2.1:** Histogram of Marks in Computer Architecture

We can create histograms for all the 4 subjects and display them in a single frame using the `par()` function as shown in the following code:

```
par(mfrow = c(2, 2))
```

```
hist(v_marks$CA, main = "Computer Architecture", xlab = "Marks")
hist(v_marks$DA, main = "Data Analysis", xlab = "Marks")
hist(v_marks$RP, main = "R Programming", xlab = "Marks")
hist(v_marks$SP, main = "Shiny Programming", xlab = "Marks")
```

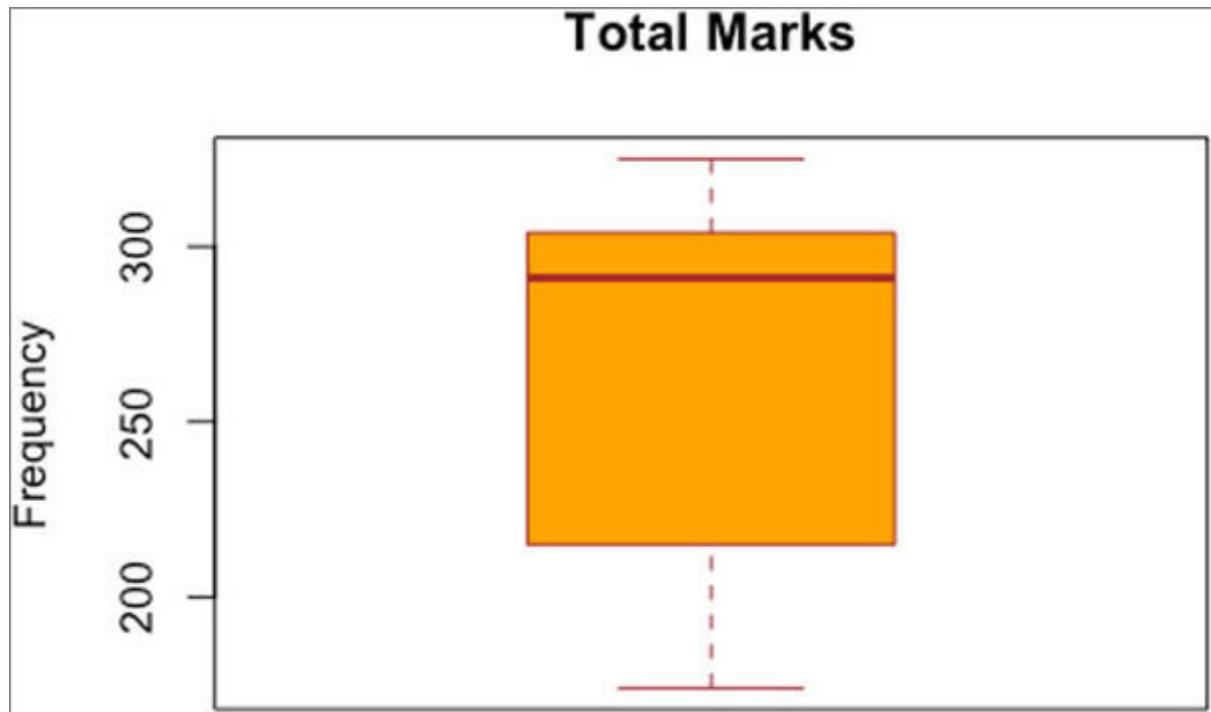


**Figure 2.2:** Histogram of all 4 subjects in one frame

## Box and Whisker chart

Box and Whisker charts are a great way to find outliers. In the Box and Whisker chart, we can visualize the Median and Quartiles as well. Let's make a Box and Whisker chart for the total marks obtained by the students as shown in the following code:

```
boxplot(v_marks$Total,  
main = "Total Marks",  
xlab = NULL,  
ylab = "Frequency",  
col = "orange",  
border = "brown",  
horizontal = FALSE,  
notch = FALSE  
)
```



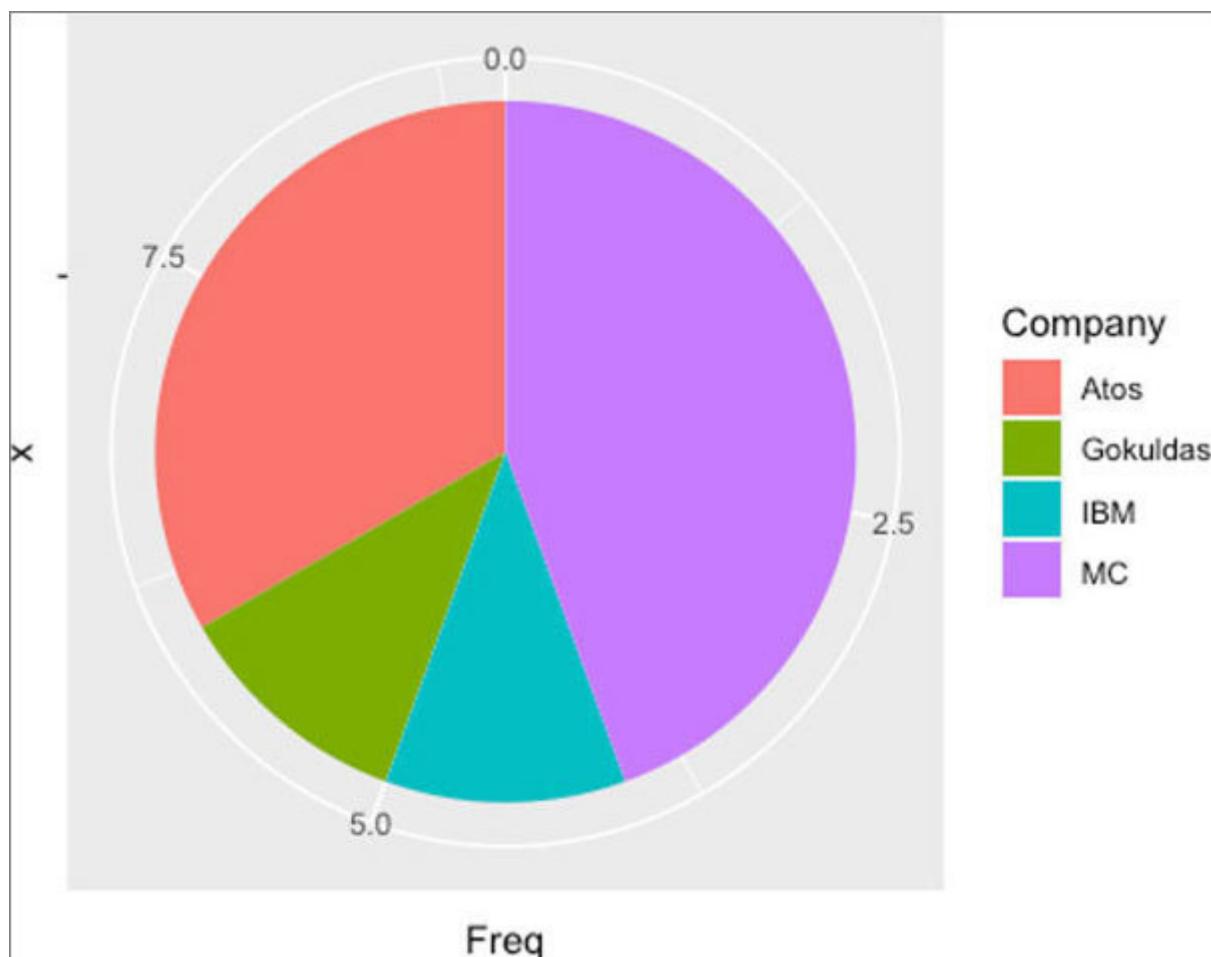
**Figure 2.3:** Box and Whisker Chart of the Total Marks

## Pie chart

Pie charts provide a visualization of the distribution. We will create pie charts using the functions in the ggplot2 package.

We will now create a pie chart of the distribution of the students in different companies as shown in the following code. Notice that before we can create the pie chart, we need to create the distribution. For creating the distribution, the table() function is used. The table created has been converted into a data frame using the as.data.frame() functions:

```
v_distribution <- with(v_marks, table(Company))
v_data <- as.data.frame(v_distribution)
v_bp <- ggplot(v_data, aes(x="", y=Freq, fill=Company)) +
  geom_bar(width = 1, stat = "identity")
v_pie <- v_bp + coord_polar("y", start=0)
v_pie
```

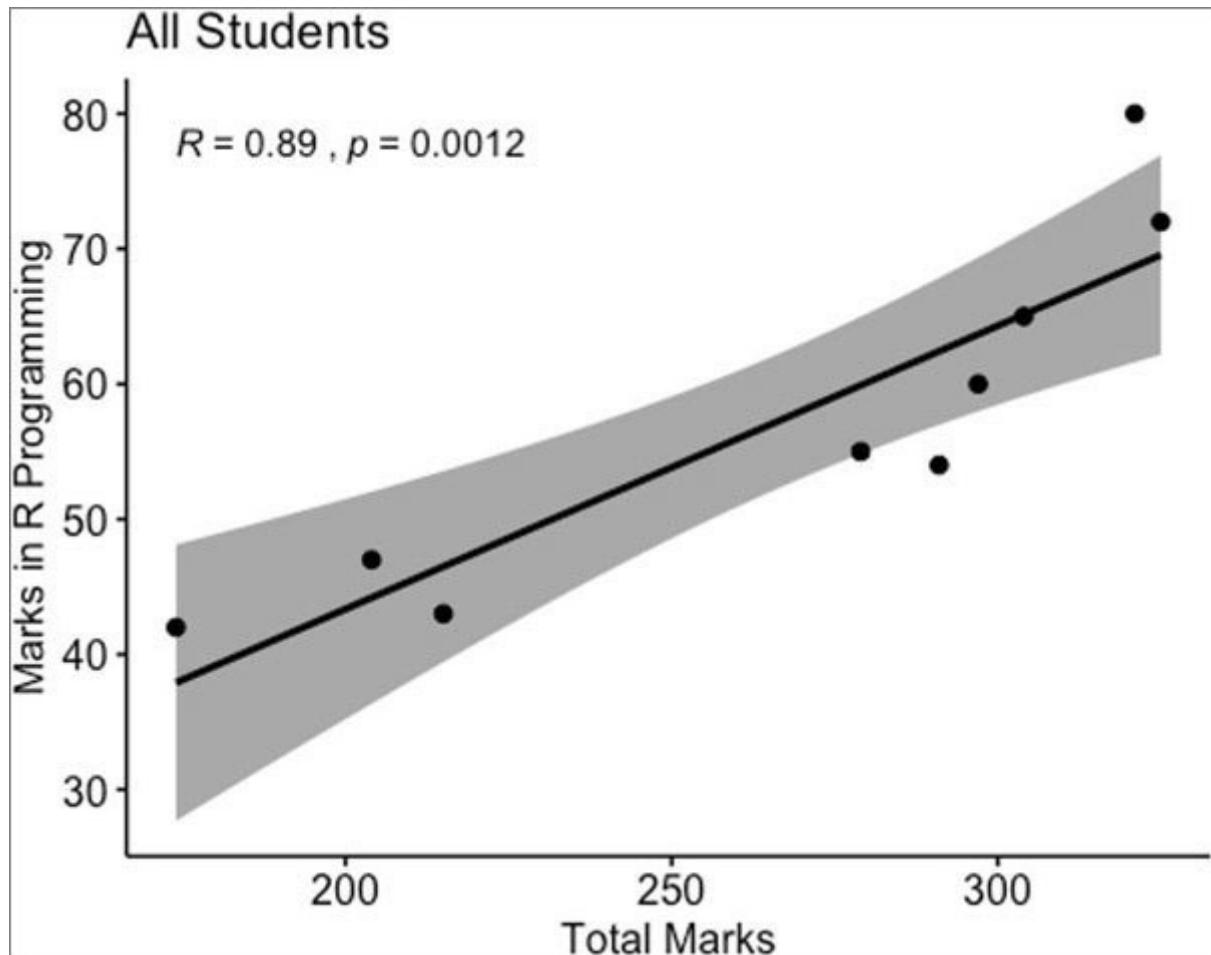


**Figure 2.4:** Pie chart of the distribution of the student by their company

## Scatter plots

Scatter plots provide a visualization of the *relationship between 2*. For creating a scatter plot, we will use the function `ggscatter()` from the library. Following code is the scatter plot between the total marks and the marks in R programming:

```
library(ggpubr)
ggscatter(v_marks,
x = "Total", y = "RP",
add = "reg.line", conf.int = TRUE,
cor.coef = TRUE, cor.method = "pearson",
xlab = "Total Marks", ylab = "Marks in R Programming",
main = 'All Students'
)
## `geom_smooth()` using formula 'y ~ x'
```



*Figure 2.5: Scatter plot*

## Reading data from a CSV file

Statistical programming is all about analyzing data. Thus, R programming always begins with getting the data to process. There are many data sets available in various R packages (or libraries). However, there is always a need for uploading data for the data sets required for our day-to-day analysis.

We generally classify data as structured data and unstructured data. Structured data has the data organized in columns and rows, like in a database. Unstructured data is free data like posts in Twitter or Facebook, or text in emails or books, etc.

In this chapter, we will discuss about structured data. Structured data can most commonly be found in **comma separated value (CSV)** files. Even the data available in Excel spreadsheets can be saved as CSV files. To upload data from CSV files, we can use the `read.csv()` function as shown in the following code:

```
v_data <- read.csv("./EmployeeJoining.csv", header = TRUE, sep =  
",", quote = "", stringsAsFactors = TRUE)  
str(v_data)  
## 'data.frame': 12333 obs. of 17 variables:  
## $ Candidate_Ref : int 2110407 2112635 2112838  
2115021 2115125 2117167 2119124 2121918 2127572 2137866 ...  
## $ DOJ_Extended : int 1 0 0 0 1 1 1 0 1 0 ...
```

```

## $ Duration_to_Accept_Offer: int 14 18 3 26 1 17 37 NA 16
NA ...
## $ Notice_Period : int 30 30 45 30 120 30 30 45
0 30 ...
## $ Offered_Band : Factor w/ 7 levels
"Eo","E1","E2",..: 3 3 3 3 3 2 3 3 2 2 ...

## $ Percent_Hike_Expected : num -20.8 50 42.8 42.8 42.6 ...
## $ Percent_Hike_Offered : num 13.2 320 42.8 42.8 42.6 ...
## $ Percent_Difference_CTC : num 42.9 180 0 0 0 ...
## $ Joining_Bonus : Factor w/ 2 levels "No","Yes": 1
1 1 1 1 1 1 1 1 ...
## $ Relocation_Status : Factor w/ 2 levels "No","Yes": 1
1 1 1 2 1 1 1 1 ...
## $ Gender : Factor w/ 2 levels
"Female","Male": 1 2 2 2 2 2 2 1 2 ...
## $ Source : Factor w/ 3 levels
"Agency","Direct",..: 1 3 1 3 3 3 3 3 2 2 ...
## $ Relevant_Experience : int 7 8 4 4 6 2 7 4 8 4 ...
## $ LOB : Factor w/ 12 levels
"AXON","BFSI",..: 7 10 10 10 10 10 10 10 9 7 ...
## $ Location : Factor w/ 12 levels
"Ahmedabad","Bangalore",..: 10 3 10 10 10 10 10 10 10 10 ...
## $ Age : int 34 34 27 34 34 34 32 34
34 34 ...
## $ Status : Factor w/ 2 levels "Joined","Not
Joined": 1 1 1 1 1 1 1 2 1 2 ...

```

We need to provide the path where the file is available. The header parameter states whether the first line of the data file contains a header or not. The sep parameter states the character

used as a separator between the fields. The quote parameter states the character used to quote the fields.

Notice that the data is read into a data frame. We see that there are more than 12,000 records in the file. So, we may want to see only a few records of the data set. We can see a few records from the top of the data set using the head() function as shown in the following code:

```
head(v_data[, 1:4])
##   Candidate_Ref DOJ_Extended Duration_to_Accept_Offer
Notice_Period
## 1      2110407          1
14      30
## 2      2112635          0
18      30
## 3
2112838          0          3          45
## 4      2115021          0
26      30
## 5
2115125          1          1          120
## 6      2117167          1
17      30
```

We can see a few records from the bottom of the data set using the tail() function as shown in the following code:

```
tail(v_data[, 1:4])
```

```

##      Candidate_Ref DOJ_Extended Duration_to_Accept_Offer
Notice_Period

## 12328      3828227          0
NA           0
## 12329      3828260          0
NA           0

## 12330      3830270          0
NA           0
## 12331
3834159          0          0          0
## 12332
3835433          0          0          30
## 12333
3836076          0          2          0

```

For finding out the number of rows in a data frame, we can use the function

```

nrow(v_data)
## [1] 12333

```

Similarly, for finding out the number of columns in a data frame, we can use the function

```

ncol(v_data)
## [1] 17

```

## Reading data from a database

Apart from files, we need reading data from databases. I present you the code for reading data from databases. We will go through this in detail later in the book.

```
library(rJava)
library(DBI)
library(RJDBC)
v_system_parameters <- read.csv(file = "SystemParameters.csv",
header = TRUE)
attach(v_system_parameters)
dsn_driver = as.character(v_system_parameters[(which(Parameter ==
"dsn_driver")),]$Value])
dsn_database = as.character(v_system_parameters[(which(Parameter ==
"dsn_database")),]$Value])
dsn_hostname = as.character(v_system_parameters[(which(Parameter ==
"dsn_hostname")),]$Value])
dsn_port = as.character(v_system_parameters[(which(Parameter ==
"dsn_port")),]$Value])
dsn_protocol = as.character(v_system_parameters[(which(Parameter ==
"dsn_protocol")),]$Value])
dsn_uid = as.character(v_system_parameters[(which(Parameter ==
"DBUserID")),]$Value])
dsn_pwd = as.character(v_system_parameters[(which(Parameter ==
"DBPassword")),]$Value])
## Connect to the Database
jcc = JDBC(dsn_driver, "Drivers/db2jcc4.jar", identifier.quote="`")
```

```

jdbc_path = paste("jdbc:db2://", dsn_hostname, ":", dsn_port, "/",
dsn_database, sep="")

v_query = paste("SELECT * FROM TB_TWEETS ",
"WHERE searchString = 'Coronavirus' ",
sep = ""
)
tryCatch(
{
conn = dbConnect(jcc, jdbc_path, user=dsn_uid,
password=dsn_pwd)
v_rs = dbSendQuery(conn, v_query);
v_temp = fetch(v_rs, -1);
if(nrow(v_temp) == 0) {
  dbDisconnect(conn)
  displayError("Error in Data Fetch from Database. ZERO ROWS
Returned. Quitting")
  stop(safeError(NULL))
}
dbDisconnect(conn)
},
error = function(e) {
# return a safeError if a parsing error occurs
dbDisconnect(conn)
displayError(paste("Error in Data Fetch from Database. Quitting",
e))
stop(safeError(e))
}
)
head(v_temp[, 1:4])
##   USER_ID
STATUS_ID          CREATED_AT      SCREEN_NAME

```

## 1	2020-02-24 03:24:21.0	davidjwhite68
## 2	2020-02-24 04:39:40.0	davidjwhite68
## 3	2020-02-24 03:19:28.0	davidjwhite68
## 4	2020-02-24 04:55:45.0	stefanoshow
## 5	2020-02-24 04:55:12.0	RebeccaLomas
## 6	2020-02-24 04:44:38.0	Muschelschloss

## Conclusion

Data frame is a data structure in R which is used very frequently in most applications written in R. For programs that we will write for emotion analysis, data frames will be used a lot data frames. So, getting familiar with data frames is of the utmost importance. This chapter has used many operations on data frames. Also, we discussed a few libraries that can make working with data frames easy. However, it will be advantageous if the learner explores more about data frames and discovers other operations are available on data frames.

Statistical programming is all about working with data. Generally, data will be found in files or database. In this chapter, we have discussed working with CSV files. However, there are other file types, like Excel spreadsheets, PDFs, Word documents, etc. from where we may have to process data. Later, in this book, we will work with data from PDF files. Getting familiar with working with data from other file types will be advantageous for the learners. The main aspect to explore is how data is loaded from different file types and how data is written to them.

The best way to persist data is by using a database. I have introduced working with databases in this chapter. We will discuss more later in the book. In the next chapter, we will discuss how to create applications in R.

### Points to remember

A data frame can be created using the `data.frame()` and the `tibble()` functions. The `tibble()` function is available in the `tibble` package.

The structure of a data frame can be obtained using the `str()` function.

To get the class of any variable in R, use the `class()` function.

Columns in a data frame can be renamed using the `rename()` function. The `rename()` function is available in the `dplyr` package.

The `which()` function can be used to get a part of a data frame based on a condition.

The `subset()` function can be used to get a subset of a data frame.

One of the methods to add a column to a data frame is by using the `cbind()` function.

One of the methods to add a row to a data frame is by using the `rbind()` function.

Data type conversion can be conducted by using the functions etc.

For sorting data in a data frame, the `arrange()` function can be used. The `arrange()` function is available in the `dplyr` package.

The `hist()` function can be used to create Histograms.

The `boxplot()` function can be used to create Box and Whisker plots.

Use the `nrow()` function to get the number of rows in a data frame.

Use the `ncol()` function to get the number of columns in a data frame.

### Multiple choice questions

Which of these is NOT unstructured data?

Collection of eMail Messages

Collection of Facebook Posts

Collection of Tweets

Daily Sensex Index for the Past 10 Years

One of the functions for setting the layout is:

display()

par()

Par()

None of the above

For checking the class of a variable in R, the function to use is:

type()

`class()`

Both A and B

None of the above

To create an empty data frame, the function to use is:

`data.frame()`

`empty.data.frame()`

`blank.data.frame()`

Empty data frames cannot be created

To create a data frame from vectors, the function to use is:

`data.frame()`

`tibble()`

Both of the above

None of the above

## Answers to MCQs

D

B

B

A

C

## Questions

Find out how to read data from an Excel spreadsheet.

Take any data from your workplace or academics. Create visualizations for this data.

Download the documentation for ggplot2 from the CRAN website and explore it.

## Key terms

**Data frame:** Data frames are widely used data structure in R. Data frames store columns of data to different data types.

**CSV:** CSV stands for comma separated value. It is a common file format for storing structured data.

**Structured data:** Data stored in rows and columns in a fixed format, like in a database, is referred to as structured data.

**Unstructured data:** Data which does not follow any fixed format is referred to as unstructured data. Examples of unstructured data includes data coming from social media.

**Vector:** Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex, or raw. A vector's type can be checked with the `typeof()` function.

**Factors:** Factors are data objects which are used to categorize data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values.

**Histogram:** A histogram is an approximate representation of the distribution of numerical or categorical data.

**Box and Whisker Chart:** A box and whisker chart shows the distribution of data into quartiles, highlighting the median and the outliers. The boxes may have lines extending vertically called These lines indicate variability outside the upper and lower quartiles, and any point outside those lines or whiskers is considered an outlier.

## CHAPTER 3

### *Developing Simple Applications in R*

So far in this book, we have been issuing individual commands to get a result. Now, we will put together the individual commands to form a program. For doing this, we will require programming constructs like conditions, loops, etc. Also, we will require writing functions in R. Let's explore all of these and more in this chapter.

R can be used for writing any kind of application. However, R is more suited and used for writing statistical applications. We will understand the construct of R programming by discussing two applications. The purpose of discussing these applications is to understand the constructs of R programming and so, the contents of the programs are less important at this stage. Later in the book, we will use these applications in the application we will write for emotion analysis.

## **Structure**

In this chapter, we will discuss the following topics:

Programming conditions and loops in R

Functions in R

Application: Number to words converter

Application: Word cloud

## Objectives

After studying this unit, you should be able to:

Understand the different programming constructs in R

Write applications using R language

## Programming conditions and loops in R

One of the essential programming constructs in any programming is to be able to make decisions based on condition(s). We have seen some aspects of this in [Chapter 2: Simple Operations Using R](#) in this book. We will elaborate the same with examples.

Conditions can be programmed in R with the if statement. Let us first discuss the syntax for the if statement:

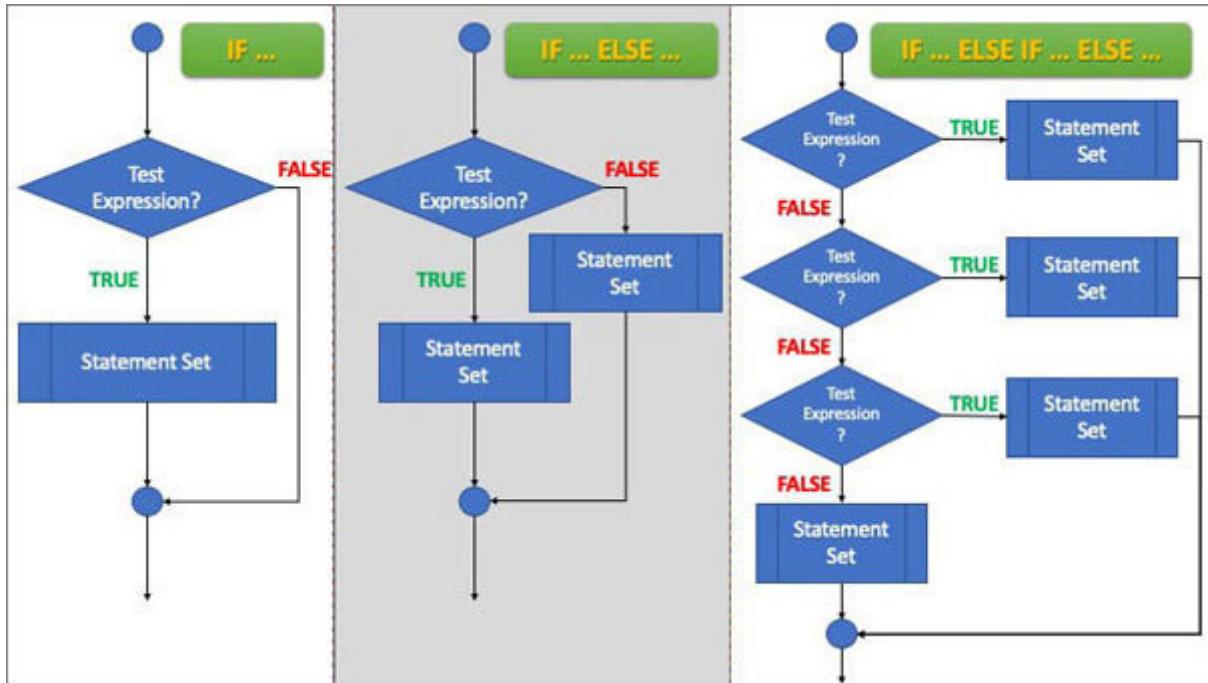
```
if(test_expression1) {  
  Statement set 1  
}
```

```
if(test_expression1) {  
  Statement set 1  
} else {  
  Statement set 2  
}
```

```
if(test_expression1) {  
  Statement set 1  
} else if(test_expresssion2) {  
  Statement set 2  
} else if(test_expresssion3) {  
  Statement set 3  
} else {
```

Statement set 4

}



*Figure 3.1: IF statement options*

We consider the same data we used in [Chapter 2: Simple Operations Using R](#) of this book for the time being.

Let us say that we need to increase the marks in R programming of the students by 10% if they belong to the company So, we have a condition to check whether the student belongs to the company MC or not. If the students belong to the company we need to increase the marks in R programming by 10%; otherwise, the marks should remain the same.

We know if the marks in a subject changes, the total marks will need to be updated. If the total marks changes, the percentage

marks will change. And if the percentage marks changes, the grade may change. So, if the marks in a subject changes, we need to update the total marks, percentage marks, and grade as well.

The existing data frame is as follows:

v\_marks

##	Name	RN	Company	CA	DA	RP	SP	Total	Percentage
Grade									
## 1	Amarendra	101	Atos	87	95	60	55	297	74.25
			B						
## 2	Anita	102	MC	86	86	65	67		
304		76.00	B						
## 3	Anuj	103	Atos	92	90	72	71	325	81.25
			A						
## 4	Arindam	104	MC	85	85	55	54		
279		69.75	C						
## 5	Deepshree	105	IBM	77	80	54	80	291	72.75
			B						
## 6	Kousik	106	Atos	90	82	80	69	321	80.25
			A						
## 7	Ranoo	108	MC	50	65	47	42		
204		51.00	D						
## 8	Babita	109	MC	45	47	42	40		
174		43.50	D						
## 9	Ujwala	110	Gokuldas	55	72	43	45	215	53.75
			D						

We can use the code given in the following code to update record number 1:

```
if(v_marks[1]$Company == "MC") {  
v_marks[1]$RP <- as.integer(v_marks[1]$RP * 1.1)  
v_marks[1]$Total<- v_marks[1]$CA + v_marks[1]$DA +  
v_marks[1]$RP + v_marks[1]$SP  
v_marks[1]$Percentage <- / 400 * 100  
  
if(v_marks[1]$Percentage > 80) {  
v_marks[1]$Grade <- "A"  
  
} else if(v_marks[1]$Percentage > 70) {  
v_marks[1]$Grade <- "B"  
} else if(v_marks[1]$Percentage > 60) {  
v_marks[1]$Grade <- "C"  
} else {  
v_marks[1]$Grade "D"  
}  
}
```

We can use the code given in the following code to update record 2:

```
if(v_marks[2]$Company == "MC") {  
v_marks[2]$RP <- as.integer(v_marks[2]$RP * 1.1)  
v_marks[2]$Total<- v_marks[2]$CA + v_marks[2]$DA +  
v_marks[2]$RP + v_marks[2]$SP  
v_marks[2]$Percentage <- v_marks[2]$Total / 400 * 100
```

```
if(v_marks[2,]$Percentage > 80) {  
v_marks[2,]$Grade <- "A"  
} else if(v_marks[2,]$Percentage > 70) {  
v_marks[2,]$Grade <- "B"  
} else if(v_marks[2,]$Percentage > 60) {  
v_marks[2,]$Grade <- "C"  
} else {  
v_marks[2,]$Grade <- "D"  
}  
}
```

After running the above two if statements, if we get the following data frame, we get as follows:

```

## 7      Ranoo 108      MC 50 65 47 42
204      51.00      D
## 8      Babita 109      MC 45 47 42 40
174      43.50      D
## 9      Ujwala 110 Gokuldas 55 72 43 45  215      53.75
D

```

Notice that there is no change in record number 1, while numbers have changed for record number 2. It should be obvious that we cannot go on like this for all the records. We would like to write a single program to run through all the records. And for that, we need the loop construct.

In R, we can program loops using the and repeat statements. We will discuss for loop and while loop as we will need them later in the book. The syntax for the for statement is provided in the following code:

```

for(val in sequence) {
  Statement set
}

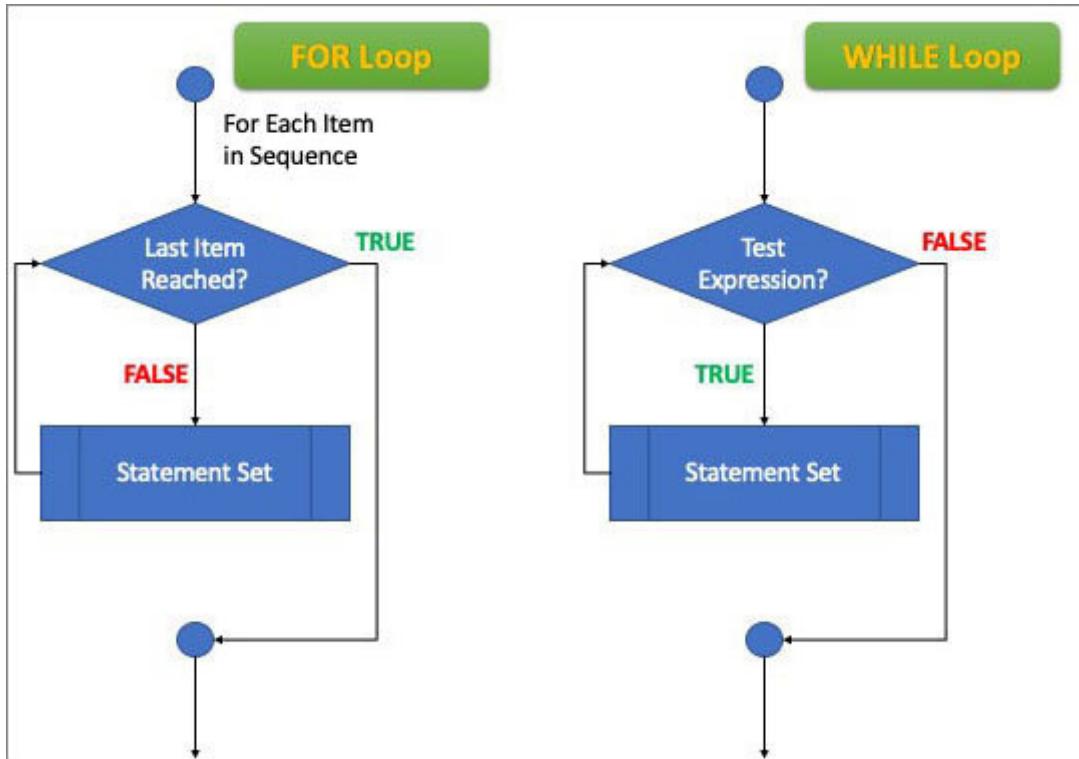
```

The syntax for the while statement is provided below:

```

while(test_expression) {
  Statement set
}

```



**Figure 3.2: FOR and WHILE statement constructs**

Using for loop, we can write our program as shown in the following code:

```

for(v_student_number in seq.int(nrow(v_marks))) {
  if(v_marks[v_student_number]$Company == "MC") {

    v_marks[v_student_number]$RP <-
      as.integer(v_marks[v_student_number]$RP * 1.1)
    v_marks[v_student_number]$Total <-
      v_marks[v_student_number]$CA + v_marks[v_student_number]$DA
      + v_marks[v_student_number]$RP + v_marks[v_student_number]$SP
    v_marks[v_student_number]$Percentage <-
      v_marks[v_student_number]$Total / 400 * 100

    if(v_marks[v_student_number]$Percentage > 80) {
  
```

```

v_marks[v_student_number]$Grade <- "A"
} else if(v_marks[v_student_number]$Percentage > 70) {
v_marks[v_student_number]$Grade <- "B"
} else if(v_marks[v_student_number]$Percentage > 60) {
v_marks[v_student_number]$Grade <- "C"
} else {
v_marks[v_student_number]$Grade <- "D"
}
}
}
}

```

The seq.int() function creates a sequence of integers. In the above program, seq.int() function creates a sequence from 1 to the number of rows in the data frame. If we see the output of the data frame now, we will find that for the students of the company the data has changed:

```

v_marks
##          Name  RN Company CA DA RP SP Total Percentage
Grade
## 1 Amarendra 101      Atos 87 95 60 55   297      74.25
B

## 2 Anita 102      MC 86 86 71 67   310      77.50
B

## 3 Anuj 103      Atos 92 90 72 71   325      81.25
A

## 4 Arindam 104      MC 85 85 60 54
284      71.00      B

## 5 Deepshree 105     IBM 77 80 54 80   291      72.75
B

```

```

## 6      Kousik 106      Atos 90 82 80 69    321      80.25
A
## 7      Ranoo 108      MC 50 65 51 42
208      52.00      D
## 8      Babita 109      MC 45 47 46 40
178      44.50      D
## 9      Ujwala 110 Gokuldas 55 72 43 45    215      53.75
D

```

Now, we conduct the same operation using the while loop. The program could be as shown in the following code:

```

v_student_number <- 1

while(v_student_number <= nrow(v_marks)) {
  if(v_marks[v_student_number,]$Company == "MC") {
    v_marks[v_student_number,]$RP <-
      as.integer(v_marks[v_student_number,]$RP * 1.1)
    v_marks[v_student_number,]$Total <-
      v_marks[v_student_number,]$CA + v_marks[v_student_number,]$DA
      + v_marks[v_student_number,]$RP + v_marks[v_student_number,]$SP

    v_marks[v_student_number,]$Percentage <-
      v_marks[v_student_number,]$Total / 400 * 100

    if(v_marks[v_student_number,]$Percentage > 80) {
      v_marks[v_student_number,]$Grade <- "A"
    } else if(v_marks[v_student_number,]$Percentage > 70) {
      v_marks[v_student_number,]$Grade <- "B"
    } else if(v_marks[v_student_number,]$Percentage > 60) {
      v_marks[v_student_number,]$Grade <- "C"
    }
  }
}

```

```
} else {  
v_marks[v_student_number]$Grade <- "D"  
}  
}  
  
v_student_number <- v_student_number + 1  
}
```

I leave it to you to check the output of the above program.

## Functions in R

While writing any application, we need pieces of code that can be used more than once in the application. The best practice is that we write such pieces of code only once and use it whenever it is required in the application. By doing this, we eliminate the duplication of code. The obvious advantage is that if we need to change such pieces of codes, then we need to change it in only one place. Another advantage is that once such pieces of codes are tested and approved, we can use them with full confidence that these code pieces will perform as required.

In R, we can write functions which encapsulate pieces of codes that can be used any number of times. We will understand functions in R through an example. In our current example of increasing marks in R programming by 10% for students of the **company** we could create a function to do this job for us as follows. We name our function as

```
IncreaseMarks <- function() {  
  v_student_number <- 1  
  while(v_student_number <= nrow(v_marks)) {  
    if(v_marks[v_student_number]$Company == "MC") {  
      v_marks[v_student_number]$RP <-  
      as.integer(v_marks[v_student_number]$RP * 1.1)  
      v_marks[v_student_number]$Total <-  
      v_marks[v_student_number]$CA + v_marks[v_student_number]$DA  
      + v_marks[v_student_number]$RP + v_marks[v_student_number]$SP
```

```

v_marks[v_student_number]$Percentage <-
v_marks[v_student_number]$Total / 400 * 100

if(v_marks[v_student_number]$Percentage > 80) {
  v_marks[v_student_number]$Grade <- "A"
} else if(v_marks[v_student_number]$Percentage > 70) {
  v_marks[v_student_number]$Grade <- "B"
} else if(v_marks[v_student_number]$Percentage > 60) {
  v_marks[v_student_number]$Grade <- "C"
} else {
  v_marks[v_student_number]$Grade <- "D"
}
}

v_student_number <- v_student_number + 1
}

return(v_marks)
}

```

To use this function, we could invoke it as shown in the following code:

```
v_marks <- IncreaseMarks()
```

If we check the data frame we notice that the marks have been increased as per our requirement:

```

v_marks
##          Name  RN Company CA DA RP SP Total Percentage
Grade

```

## 1	Amarendra 101	Atos	87 95 60 55	297	74.25
B					
## 2	Anita 102	MC	86 86 71 67	310	77.50
B					
## 3	Anuj 103	Atos	92 90 72 71	325	81.25
A					
## 4	Arindam 104	MC	85 85 60 54		
284	71.00	B			
## 5	Deepshree 105	IBM	77 80 54 80	291	72.75
B					
## 6	Kousik 106	Atos	90 82 80 69	321	80.25
A					
## 7	Ranoo 108	MC	50 65 51 42		
208	52.00	D			
## 8	Babita 109	MC	45 47 46 40		
178	44.50	D			
## 9	Ujwala 110	Gokuldas	55 72 43 45	215	53.75
D					

However, this function is not very useful. It will be a one-time requirement that we will increase marks for a subject for a company by a fixed amount. A slightly more useful requirement could be if we could state the company for which the marks in R programming have to be increased and by how much. To do this, we could pass parameters to the function. For this requirement, we could pass a parameter for the company for which marks in R programming have to be increased. And also provide a second parameter stating the percentage increase to apply for the marks in R programming.

The program could be altered as shown in the following code:

```
IncreaseMarks <- function(p_company, p_percent_to_increase) {  
  v_student_number <- 1  
  while(v_student_number <= nrow(v_marks)) {  
  
    if(v_marks[v_student_number,]$Company == p_company) {  
      v_marks[v_student_number,]$RP <-  
        as.integer(v_marks[v_student_number,]$RP * (1 +  
          p_percent_to_increase))  
      v_marks[v_student_number,]$Total <-  
        v_marks[v_student_number,]$CA + v_marks[v_student_number,]$DA  
        + v_marks[v_student_number,]$RP + v_marks[v_student_number,]$SP  
      v_marks[v_student_number,]$Percentage <-  
        v_marks[v_student_number,]$Total / 400 * 100  
  
      if(v_marks[v_student_number,]$Percentage > 80) {  
        v_marks[v_student_number,]$Grade <- "A"  
      } else if(v_marks[v_student_number,]$Percentage > 70) {  
        v_marks[v_student_number,]$Grade <- "B"  
      } else if(v_marks[v_student_number,]$Percentage > 60) {  
        v_marks[v_student_number,]$Grade <- "C"  
      } else {  
        v_marks[v_student_number,]$Grade <- "D"  
      }  
    }  
  
    v_student_number <- v_student_number + 1  
  }  
  return(v_marks)  
}
```

We could invoke this function as shown in the following code. I have shown one example of how parameters could be passed. We could provide different values for both the parameters:

```
v_marks <- IncreaseMarks("Atos", 0.05)
```

I leave it to you to verify the output.

## *Application: number to words converter*

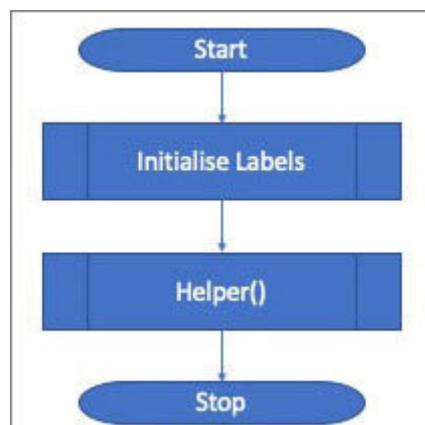
We have gone through all the elements required to create applications in R. So, let us now create our first application in R. The first application that we discuss is to convert numbers to words. Given a number, like 11, we need to generate it in words, that is, eleven.

This application has a very wide usage. For example, anywhere that an amount is involved in an invoice or it bill; the amount must be stated in both numbers and words. The amount of the invoice is computed after accumulating all the involved elements. Once we have the figure, the amount needs expressing in words as well. This is a mandatory requirement.

## Writing our own function

Our purpose is that we need a function to which if we input a number, the function should return us the number in words. In [Chapter 5: Shiny Application](#) we will discuss how to create a web application serving the same purpose. We go through the logic that we will implement. In parallel, we will see the associated code. In the end, we will put together all the pieces of the code to form the application.

The logic for the main function will be as shown in the following figure:



**Figure 3.3:** Number to words - Main Function Logic

We need to initialize the labels. Labels are words associated with numbers. For example, we need the label for all the digits between 0 and 9, that is, zero, one, two, etc. After that, we will

call a helper function where the logic for converting numbers to words will reside. The code is as shown in the following code:

```
numbers2wordsIndia <- function(x){  
  #Disable scientific notation  
  options(scipen = 999)  
  
  ones <- c("zero", "one", "two", "three", "four", "five", "six", "seven",  
  "eight", "nine")  
  names(ones) <- 0:9  
  teens <- c("ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen",  
  "sixteen", "seventeen", "eighteen", "nineteen")  
  <- 0:9  
  tens <- c("twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty",  
  "ninety")  
  names(tens) <- 2:9  
  x <- round(x)  
  if (length(x) > 1) return(trim(sapply(x, helper)))  
  helper(x)  
}
```

Note that we have named our function as This is because we will discuss the function for converting numbers to words as per the Indian convention. In the Indian convention, (from the right) the first 3 digits are clubbed till hundreds and then every 2 digits are clubbed. This is unlike the European convention where every 3 digits are clubbed from the right.

We first disable scientific representation for numbers. We do this to allow for the input of very large numbers. The code for

achieving this is provided in the following code:

```
options(scipen = 999)
```

Next, we declare 3 vectors – ones, teens, and tens. These 3 vectors store the labels for numbers 0-9, 11-19, and all the tens (that is, 10, 20, 30, etc.). names() function assigns column names. We have assigned the column names according to the position of the represented number. I present one example below to explain the same:

```
ones <- c("zero", "one", "two", "three", "four", "five", "six", "seven",
"eight", "nine")
names(ones) <- 0:9
```

```
ones
## 0 1 2 3 4 5 6 7 8 9
## "zero" "one" "two" "three" "four" "five" "six" "seven" "eight"
"nine"
```

So, ones[0] will return zero and so on. We use the round() function for ensuring that we only deal with integers. Next, note this line of code as shown in the following code:

```
if (length(x) > 1 && x >= 0) return(trim(sapply(x, helper)))
```

We know that the length() function returns the number of elements in the vector. This piece of code has been included so that the function can also handle a vector of numbers instead of

just a single number. The function `sapply()` applies the same function, in this case the function on each element of the vector

## trim() function

We take a slight diversion and complete the discussion on the trim() function here. The trim() function is used to remove unnecessary elements from the word formed from the conversion. The code is provided in the following code:

```
trim <- function(text){  
  #Tidy leading/trailing whitespace, space before comma  
  text <- gsub("^\\ ", "", gsub("\\ *$", "", gsub("\\,", ",", text)))  
  #Clear any trailing " and"  
  gsub(" and$", "", text)  
}
```

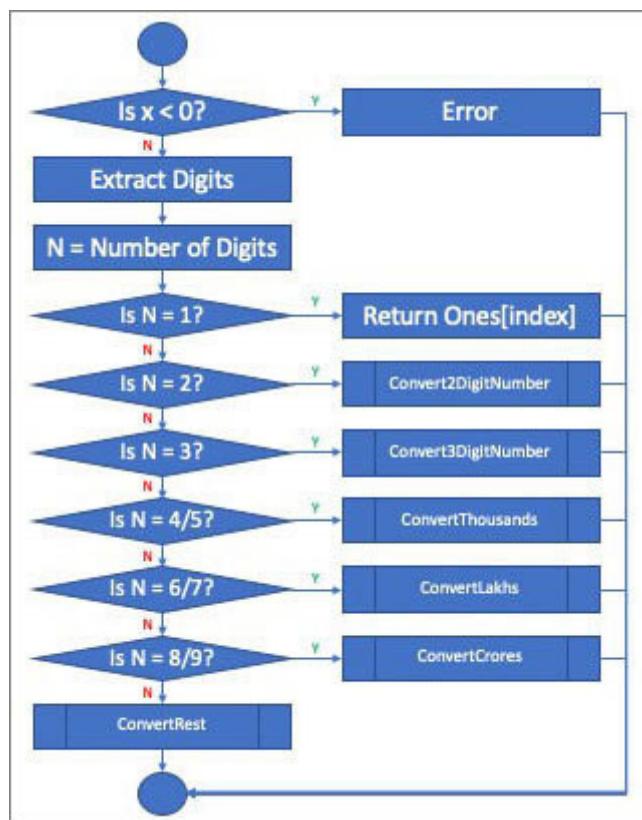
The gsub() function is used for the substitution of patterns using a regular expression. The syntax for the gsub() function is as shown in the following code:

```
gsub(to search>, to replace by>, to scan>)
```

In regular expressions, ^ indicates beginning of line, \$ indicates end of line. Suggest reading any material on regular expressions to understand the patterns.

## helper() function

Next, we discuss the helper() function. The flow chart is shown in the following figure:



**Figure 3.4:** Number to words - Helper Function Logic

As you can see in the flow chart, the helper() function only converts the positive integers from numbers to words. If any number less than 0 is provided, the function should return an error.

We first extract the digits from the provided number and count the number of digits in the provided number. Based on the number of digits in the provided number, we will use different functions for converting the number from numbers to words. Notice that I have planned separate functions for converting 2-digit numbers, 3-digit numbers, numbers in thousands, numbers in lakhs, numbers in crores, and any number larger than that.

The associated code is as shown in the following code:

```
helper <- function(x) {  
  if (x < 0) {return(paste(x, "is negative!"))}  
  
  digits <- rev(strsplit(as.character(x), "")[[1]])  
  nDigits <- length(digits)  
  if (nDigits == 1) as.vector(ones[digits])  
  else if (nDigits == 2)  
    convert2DigitNumbers(x)  
  else if (nDigits == 3)  
    convert3DigitNumbers(x)  
  else if (nDigits == 4 || nDigits == 5)  
    convertThousands(x)  
  else if (nDigits == 6 || nDigits == 7)  
    convertLakhs(x)  
  else if (nDigits == 8 || nDigits == 9)  
    convertCrores(x)  
  else  
  
  trim(paste(numbers2wordsIndia(floor(x/10000000)), "crore",  
            convertCrores(x %% 10000000)))
```

```
}
```

**Note that this is a recursive function, i.e. the function is called from within the function itself. We use recursion for handling numbers larger than crores.**

This code should be easy to understand as most of it is a ladder of if statements. Let us go through the code for extracting the digits from the provided number. The code we want to discuss is shown in the following code:

```
digits <- rev(strsplit(as.character(x), "")[[1]])
```

To understand this code, let us run it piece by piece as shown below:

```
strsplit(as.character(12345), "")  
## [[1]]  
## [1] "1" "2" "3" "4" "5"
```

```
strsplit(as.character(12345), "")[[1]]  
## [1] "1" "2" "3" "4" "5"
```

```
rev(strsplit(as.character(12345), "")[[1]])  
## [1] "5" "4" "3" "2" "1"
```

We know that the `as.character()` function converts a number to a string. We use the `strsplit()` function for separating each digit as an individual character (notice that we used an empty string as a separator so that each digit gets separated). However, the

`strsplit()` function returns an array of vectors with 1 vector in it. So, we extract out this vector. Then, we apply the `rev()` function to reverse the order of the digits.

As we have a vector now, we can use the `length()` function to determine the number of digits in the number. Based on the number of digits in the number, we can decide which function to use for converting the number to words.

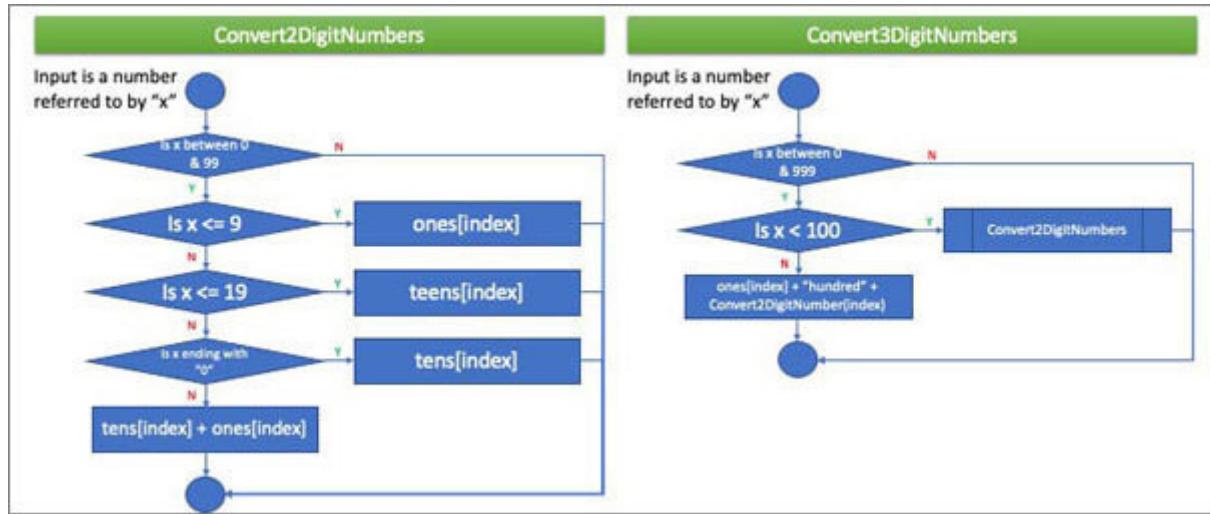
Before we move to the individual functions for converting a number to words, let us discuss the `paste()` function. The `paste()` function is used to concatenate 2 or more elements. The `paste()` function is essential when we need to concatenate the elements of different data types. Let us examine this statement in the following code:

```
paste(x, "is negative")
```

Here, `x` is a numeric variable, and it needs concatenating with a string. Suppose the value of `x` is the output of the statement will be as shown in the following code:

```
paste(-12345, "is negative")
## [1] "-12345 is negative"
```

Now, we discuss the functions for converting numbers of different lengths to words. We start with the functions `convert2DigitNumbers()` and



**Figure 3.5:** Convert function logic – for 2-digit and 3-digit numbers

## convert2DigitNumbers() function

First, we discuss the code for the convert2DigitNumbers() function:

```
convert2DigitNumbers <- function(x) {  
  if (x > 0 && x <= 99) {  
    digits2DigitNumber <- rev(strsplit(as.character(x), "")[[1]])  
    if (x <= 9) as.vector(ones[digits2DigitNumber])  
    else if (x <= 19) as.vector(teens[digits2DigitNumber[1]])  
    else {  
      if (digits2DigitNumber[1][1] == "0") {  
        trim(paste(tens[digits2DigitNumber[2]]))  
      }  
      else {  
        trim(paste(tens[digits2DigitNumber[2]],  
                  as.vector(ones[digits2DigitNumber[1]])))  
      }  
    }  
  }  
}
```

This code should be understandable as we have discussed all the elements before. Let us discuss the variable digits2DigitNumber by studying its value as shown in the following code:

```
digits2DigitNumber <- rev(strsplit(as.character(56), "")[[1]])
```

```
digits2DigitNumber  
## [1] "6" "5"  
digits2DigitNumber[1]  
## [1] "6"
```

```
digits2DigitNumber[1][1]
```

```
## [1] "6"
```

```
tens[digits2DigitNumber[2]]  
##      5  
## "fifty"
```

```
ones[digits2DigitNumber[1]]  
##      6  
## "six"
```

This illustration should make it clear how we convert the digits to word. We can test this function as shown in the following code:

```
convert2DigitNumbers(22)  
## [1] "twenty two"  
convert2DigitNumbers(8)  
## [1] "eight"  
convert2DigitNumbers(16)  
## [1] "sixteen"
```

### convert3DigitNumbers() function

The code for convert3DigitNumbers() function is provided in the following code:

```
convert3DigitNumbers <- function(x) {  
  if (x > 0 && x <= 999) {  
    if (x < 100) convert2DigitNumbers(x)  
    else {  
      digits3DigitNumber <- rev(strsplit(as.character(x), "")[[1]])  
      trim(paste(ones[digits3DigitNumber[3]], "hundred and",  
                convert2DigitNumbers(makeNumber(digits3DigitNumber[2:1]))))  
    }  
  }  
}
```

You would notice a new representation as Let us see the output of this code for better clarity and then it should be clear:

```
digits3DigitNumber <- rev(strsplit(as.character(176), "")[[1]])  
digits3DigitNumber[2:1]  
## [1] "7" "6"
```

This code should be understandable now. However, we make use of a new function Let us see the code for this function and discuss it:

```
makeNumber <- function(...) as.numeric(paste(..., collapse=""))
```

The makeNumber() function accepts any number of arguments. This is the purpose of using ... as the parameter. We input a vector of characters to Each character in the input is a digit. We use the paste() function to concatenate these characters into a string without any character between these individual digits. This forms a number as a string. We convert this string into a number using the as.numeric() function.

See the example in the following code:

```
digits3DigitNumber <- rev(strsplit(as.character(176), "")[[1]])  
digits3DigitNumber[2:1]  
## [1] "7" "6"
```

```
makeNumber(digits3DigitNumber[2:1])  
## [1] 76
```

I will perform one test on the function convert3DigitNumbers() and leave it to you for further testing:

```
convert3DigitNumbers(872)  
## [1] "eight hundred and seventy two"
```

We have discussed all the programming constructs that we require. So, let us study the code required for the functions and convertCrores() and perform some tests on them. I would encourage you to conduct thorough tests on each of these

functions to understand them better and ensure that they work as required.

## convertThousands() function

Shown below is the code for the function

```
convertThousands <- function(x) {  
  if (x > 0 && x <= 99999) {  
    if (x < 1000) convert3DigitNumbers(x)  
    else {  
      digitsThousands <- rev(strsplit(as.character(x), "")[[1]])  
      if (x <= 9999)  
        trim(paste(ones[digitsThousands[4]], "thousand",  
                  convert3DigitNumbers(makeNumber(digitsThousands[3:1]))))  
      else  
        trim(paste(convert2DigitNumbers(makeNumber(digitsThousands[5:4])),  
                  "thousand",  
                  convert3DigitNumbers(makeNumber(digitsThousands[3:1]))))  
    }  
  }  
}
```

Following code are the test results for the function

```
convertThousands(1000)  
## [1] "one thousand"
```

```
convertThousands(100)  
## [1] "one hundred"
```

```
convertThousands(7156)
## [1] "seven thousand one hundred and fifty six"
```

```
convertThousands(16590)
## [1] "sixteen thousand five hundred and ninety"
```

```
convertThousands(34912)
## [1] "thirty four thousand nine hundred and twelve"
```

## convertLakhs() function

Following code is the code for the function

```
convertLakhs <- function(x) {  
  if (x > 0 && x <= 9999999) {  
    if (x < 100000) convertThousands(x)  
    else {  
      digitsLakhs <- rev(strsplit(as.character(x), ""))[[1]]  
      if (x <= 99999)  
        "lakh", convertThousands(makeNumber(digitsLakhs[5:1])))  
      else  
        trim(paste(convert2DigitNumbers(makeNumber(digitsLakhs[7:6])),  
                  "lakh", convertThousands(makeNumber(digitsLakhs[5:1]))))  
    }  
  }  
}
```

Following code are the test results for the function

```
convertLakhs(100000)  
## [1] "one lakh"
```

```
convertLakhs(1000000)  
## [1] "ten lakh"
```

```
convertLakhs(715652)
```

```
## [1] "seven lakh fifteen thousand six hundred and fifty two"
```

```
convertLakhs(659001)
```

```
## [1] "six lakh fifty nine thousand one"
```

```
convertLakhs(4912886)
```

```
## [1] "forty nine lakh twelve thousand eight hundred and eighty six"
```

## convertCrores() function

Following code is the code for the function

```
convertCrores <- function(x) {  
  if (x > 0 && x <= 999999999) {  
    if (x < 10000000) convertLakhs(x)  
    else {  
      digitsCrores <- rev(strsplit(as.character(x), ""))[[1]]  
      if (x <= 9999999)  
        trim(paste(ones[digitsCrores[8]], "crore",  
                  convertLakhs(makeNumber(digitsCrores[7:1]))))  
      else  
        trim(paste(convert2DigitNumbers(makeNumber(digitsCrores[9:8])),  
                  "crore", convertLakhs(makeNumber(digitsCrores[7:1]))))  
    }  
  }  
}
```

Following code are the test results for the function

```
convertCrores(10000000)  
## [1] "one crore"
```

```
convertCrores(100000080)  
## [1] "ten crore eighty"
```

```
convertCrores(71565261)
## [1] "seven crore fifteen lakh sixty five thousand two hundred
and sixty one"
```

```
convertCrores(659001003)
## [1] "sixty five crore ninety lakh one thousand three"
```

```
convertCrores(491288637)
## [1] "forty nine crore twelve lakh eighty eight thousand six
hundred and thirty seven"
```

## [number2wordsIndia\(\) function](#)

Now, we have all the parts we need for the number2wordsIndia() function. Let us put them together. Notice that we have several functions required to create this application. We use a feature of R where we can declare functions inside the body of a function. When we do so, the functions declared inside the body of the parent function become available only within the functions and they cannot be called from outside. However, this makes our parent function fully self-contained. This feature is very important for making applications:

Following code for the function

```
numbers2wordsIndia <- function(x) {  
  helper <- function(x) {  
    if (x < 0) {return(paste(x, "is negative!"))}  
  
    digits <- rev(strsplit(as.character(x), "")[[1]])  
    nDigits <- length(digits)  
    if (nDigits == 1) as.vector(ones[digits])  
    else if (nDigits == 2)  
      convert2DigitNumbers(x)  
    else if (nDigits == 3)  
      convert3DigitNumbers(x)  
    else if (nDigits == 4 || nDigits == 5)  
      convertThousands(x)
```

```
else if (nDigits == 6 || nDigits == 7)
convertLakhs(x)
else if (nDigits == 8 || nDigits == 9)
convertCrores(x)
else

trim(paste(numbers2wordsIndia(floor(x/10000000)), "crore",
convertCrores(x %% 10000000)))
}
```

```
convert2DigitNumbers <- function(x) {
if (x > 0 && x <= 99) {
digits2DigitNumber <- rev(strsplit(as.character(x), "")[[1]])
if (x <= 9) as.vector(ones[digits2DigitNumber])
else if (x <= 19) as.vector(teens[digits2DigitNumber[1]])
else {
if (digits2DigitNumber[1][1] == "o") {
trim(paste(tens[digits2DigitNumber[2]]))
}
else {
trim(paste(tens[digits2DigitNumber[2]],
as.vector(ones[digits2DigitNumber[1]])))
}
}
}
}
}
}
```

```
convert3DigitNumbers <- function(x) {
if (x > 0 && x <= 999) {
if (x < 100) convert2DigitNumbers(x)
else {
```

```
    digits3DigitNumber "")[[1]])  
    trim(paste(ones[digits3DigitNumber[3]], "hundred and",  
    convert2DigitNumbers(makeNumber(digits3DigitNumber[2:1]))))  
  }  
}  
}
```

```
convertThousands <- function(x) {  
  if (x > 0 && x <= 99999) {  
    if (x < 1000) convert3DigitNumbers(x)  
    else {  
      digitsThousands <- rev(strsplit(as.character(x), "")[[1]])  
      if (x <= 9999)  
        trim(paste(ones[digitsThousands[4]], "thousand",  
        convert3DigitNumbers(makeNumber(digitsThousands[3:1]))))  
      else  
        trim(paste(convert2DigitNumbers(makeNumber(digitsThousands[5:4])),  
        "thousand",  
        convert3DigitNumbers(makeNumber(digitsThousands[3:1]))))  
    }  
  }  
}  
}
```

```
convertLakhs <- function(x) {  
  if (x > 0 && x <= 9999999) {  
    if (x < 100000) convertThousands(x)  
    else {  
      digitsLakhs <- rev(strsplit(as.character(x), "")[[1]])  
      if (x <= 999999)
```

```

trim(paste(ones[digitsLakhs[6]], "lakh",
convertThousands(makeNumber(digitsLakhs[5:1]))))
else

trim(paste(convert2DigitNumbers(makeNumber(digitsLakhs[7:6])),
"lakh", convertThousands(makeNumber(digitsLakhs[5:1]))))
}
}
}

```

```

convertCrores <- function(x) {
if (x > 0 && x <= 999999999) {
if (x < 10000000) convertLakhs(x)
else {
digitsCrores <- rev(strsplit(as.character(x), "")[[1]])
if (x <= 9999999)
trim(paste(ones[digitsCrores[8]], "crore",
convertLakhs(makeNumber(digitsCrores[7:1]))))
else
trim(paste(convert2DigitNumbers(makeNumber(digitsCrores[9:8])),
"crore", convertLakhs(makeNumber(digitsCrores[7:1]))))
}
}
}

```

```

trim <- function(text){
#Tidy leading/trailing whitespace, space before comma
text=gsub("^\\ ", "", gsub("\\ *$", "", gsub("\,","",text)))

```

```

#Clear any trailing " and"
gsub(" and$", "", text)

```

```
}
```

```
makeNumber <- function(...) as.numeric(paste(..., collapse=""))
```

```
#Disable scientific notation  
options(scipen=999)
```

```
ones <- c("zero", "one", "two", "three", "four", "five", "six", "seven",  
"eight", "nine")  
names(ones) <- 0:9
```

```
teens <- c("ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen",  
"sixteen", "seventeen", "eighteen", "nineteen")  
names(teens) <- 0:9
```

```
tens <- c("twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty",  
"ninety")  
names(tens) <- 2:9
```

```
x <- round(x)  
if (length(x) > 1) return(trim(sapply(x, helper)))
```

```
helper(x)  
}
```

Following code are some test results for the function

```
numbers2wordsIndia(o)
```

```
## [1] "zero"

numbers2wordsIndia(6)
## [1] "six"

numbers2wordsIndia(15)
## [1] "fifteen"

numbers2wordsIndia(107)
## [1] "one hundred and seven"

numbers2wordsIndia(1895321)
## [1] "eighteen lakh ninety five thousand three hundred and
twenty one"

numbers2wordsIndia(1297538954118909)
## [1] "twelve crore ninety seven lakh fifty three thousand eight
hundred and ninety five crore forty one lakh eighteen thousand
nine hundred and nine"

# Vector as Input
numbers2wordsIndia(c(12700, 854327, 100, 76346822))
## [1] "twelve thousand seven hundred"
## [2] "eight lakh fifty four thousand three hundred and twenty
seven"
## [3] "one hundred"
## [4] "seven crore sixty three lakh forty six thousand eight
hundred and twenty two"
```

When using the `number2wordsIndia()` function with very large numbers, you need to be aware that the precision of the large numbers may be lost due to the capacity of the machine on which this function is run. You may notice that the words generated for the last few digits of such large numbers are different from the digits of the number provided. See an example in the following code:

```
numbers2wordsIndia(1297538954118909541)
## [1] "twelve thousand nine hundred and seventy five crore thirty
eight lakh ninety five thousand four hundred and eleven crore
eighty nine lakh nine thousand four hundred and forty"
```

## R library for number to words conversion

R provides the library english which has the function english() for converting numbers to words. The english() function does the conversion as per the European convention. In European convention, we have units, tens, hundreds, thousands, millions, billions, and trillions.

Shown below is the usage of the english() function. Note the syntax of using We need this syntax to make sure that the english() function is used from the english library. There are many functions with the same names from different libraries. Each of such functions may have been created for a different purpose and may need different set of parameters. In such cases, if the proper library is not pointed to, the program may not compile and/or may not work properly:

```
english::english(0)
## [1] zero
```

```
english::english(19)
## [1] nineteen
```

```
english::english(173)
## [1] one hundred seventy-three
```

```
english::english(1234)
## [1] one thousand two hundred thirty-four
```

```
english::english(1234876438)
## [1] one billion two hundred thirty-four million eight hundred
seventy-six thousand four hundred thirty-eight
```

```
english::english(771234876438)
## [1] seven hundred seventy-one billion two hundred thirty-four
million eight hundred seventy-six thousand four hundred thirty-
eight
```

## Application: Word Cloud

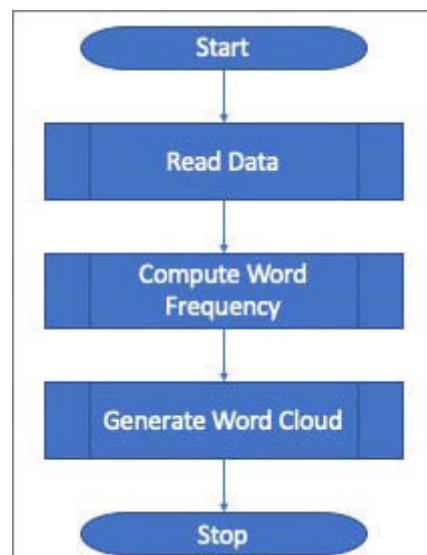
We will now discuss an application from the area of text mining. In most websites, you would have noticed Word Clouds. Word Clouds provide an effective visualization of the main contents of the discussion in any text. The idea behind a Word Cloud is to compute the Word Frequency in any text. Then we display the words of the text such that the words used more in the text are displayed in a more prominent manner. This way we get to see what the main topic/topic is/are being discussed in the text. For example, for any blog, if we create a Word Cloud, we can provide a visualization of the topics that the blog mainly discusses.

We will see how to create Word Clouds from structured and unstructured data.

By structured data, we refer to the data which is organized in some order. For example, the data in a database is arranged in rows and columns. Every table in a database has defined columns and the data in each column follows the structure and the rules imposed on that column.

In contrast, unstructured data follows no pattern. We can see examples of unstructured data in data coming from social media like Facebook or Twitter, etc. Or for that matter, if we take the contents of a book or any PDF file or any text file, the data we will get will not follow any regular pattern.

The general logic for writing this application is provided in the following figure:



**Figure 3.6:** Program logic for generating a Word Cloud

## Creating a Word Cloud from structured data

The data I use for discussing the Word Cloud based on structured data is the data regarding my expenses that I track daily. Now, what I track is the expense head, date of expense, and the amount spent. From these fields, we can create a Word Cloud on the data in the field expense head to see what the main expenditures are that I incur in terms of the frequency of the times I spend on that expense head. This data is being taken from a CSV file for this discussion.

The first step is to read the data from the CSV file. The code is as shown in the following code :

```
v_data <- read.csv("./Expenses.csv", sep = ",")
```

We can view the data frame v\_data as shown in the following code:

```
head(v_data, 10)
##          Date      Narration Amount Category
## 1 01-Oct-17           Tea      7   Luxury
## 2 01-Oct-17 Mutual Fund  5000  Savings
## 3 01-Oct-17 Bakshis to Security  100 Donation
## 4 01-Oct-17           Tea      7   Luxury
## 5 01-Oct-17      Fruits    270    Food
## 6 01-Oct-17     Envelops   240 Essential
```

```

## 7 01-Oct-17          Vegetables    100     Food
## 8 01-Oct-17          Tea        7     Luxury
## 9 01-Oct-17          Vegetables    15     Food
## 10 01-Oct-17 Bakshis to Security 100   Donation

```

Note that, besides the data frame, we passed an additional parameter to the `head()` function. To the `head()` function, we can state the number of rows to display from the top. We see that the expense head is stored in the column So, we compute the frequency of the individual items in the column Narration as shown in the following code:

```

library("plyr")
v_counts <- count(v_data, 'Narration')

```

We have used the `count()` function from the library The frequency table is stored in the data frame named

We can view the contents of the data frame `v_counts` as shown in the following code:

```

head(v_counts, 10)
##                                     Narration freq
## 1                         AC Repair      2
## 2                     Adaptor       3
## 3                   Affidavit      2
## 4                  Air Ticket     33
## 5                   Alcohol      11
## 6                      Alm        1

```

```
## 7           Animal Care    2
## 8           Aquarium Fish   5
## 9 Armed Forces Flag Day Fund  2
## 10          Atta            2
```

From this frequency table, we can generate the Word Cloud using the wordcloud() function from the wordcloud library as shown in the following code:

```
library("wordcloud")
```

```
## Loading required package: RColorBrewer
```

```
wordcloud(words = v_counts$Narration, freq = v_counts$freq,
min.freq = 1, max.words=250,
colors = brewer.pal(8, "Dark2"),
random.order = FALSE)
```



**Figure 3.7**

## [Creating a Word Cloud from unstructured data](#)

The data I use for discussing the Word Cloud based on unstructured data is the contents of my resume. The resume has been stored as a text (TXT) file.

For this program, we will need several libraries. So, we will load the libraries first. I would encourage you to search the CRAN website to collect the documentations for these libraries and study them in details:

```
library(tibble)
library(readr)
library(textclean)
library(tm)
## Loading required package: NLP
```

Given below is the function we will use to calculate the word frequency:

```
calculateWordFrequency <- function(l_text) {
  # Create the Corpus
  v_corpus <- Corpus(VectorSource(l_text))

  # Create the Term Document Matrix after cleaning the Corpus
  v_tdm <- TermDocumentMatrix(v_corpus, control =
    list(removePunctuation = TRUE,
```

```

stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf))
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))

return (v_data)
}

```

There are lots of elements of Text Miningtext mining in the above code. Let us go through them. When dealing with raw text, we need to form a Corpus. A Corpus is the singular for the term Corpora. Corpora are collections of documents containing (natural language) text. The function Corpus() from the library tm is used to form a corpus. In the function presented above, we pass the text as vector to the function. So, we use the function VectorSource() to pass the vector to the function Corpus() for forming the corpus.

From the Corpus, we can create the Term Document Matrix using the `TermDocumentMatrix()` function. The Term Document Matrix will contain all the terms (words) contained in the Corpus.

However, while forming the Term Document Matrix, we remove the terms that we do not need. In this function, we have removed all the punctuation marks, stop words (words like and, or, the, etc.), numbers, etc. Also, we have converted all the words to the lower case.

From the Term Document Matrix, we form the word frequency using the function `WordFrequency()`. While using the function we can specify the frequency range of the words to retain. Here, we have specified 1 to Infinity to include all the words in the corpus. Once we have the word frequencies, we convert it into a matrix using the function `matrix()`. This is an intermediate step so that we can perform computations on the Word Frequencies.

In the last step, we sort the terms in the descending order of their frequencies and convert the result to a data frame using the function `as.data.frame()`. Now that we have our function for computing the word frequency, we will proceed to create the Word Cloud. The process is the same as before. We will first load the data and compute the frequency table as shown below:

```
v_text <- tibble(text = gsub("[^[:alnum:]]//'", "",  
read_lines("./Resume.txt")))  
# Drop the Empty Lines  
v_raw_text <- drop_empty_row(v_text)
```

```
# Calculate Word Frequency  
v_frequency_table <- calculateWordFrequency(v_raw_text)
```

We have used the function **read\_lines()** from the library `readr` to read the text file. **drop\_empty\_row()** removes all the empty lines read from the file. The computed Word Frequencies are stored in the data frame

Let us see the contents of this data frame:

```
head(v_frequency_table, 10)  
##   values      ind  
## 1     71 develop  
## 2     61 system  
## 3     51 manag  
## 4     49 use  
## 5     34 solut  
## 6     33 project  
## 7     25 creat  
## 8     25 saudi  
## 9     24 arabia  
## 10    23 data
```

From this frequency table, we can form the Word Cloud as shown in the following code:

```
wordcloud(words = v_frequency_table$ind, freq =  
v_frequency_table$values,
```

```

min.freq = 1, max.words = 250,
random.order = FALSE,
colors=brewer.pal(8, "Dark2")
)

```



*Figure 3.8*

In [Chapter 6: Shiny Application](#) we will write a web application to create a Word Cloud for any file provided as input.



## Conclusion

Billions of applications have already been developed using R language. If one searches the available R libraries, chances are that one would find a library containing the application one needs. However, these libraries do not provide solutions required for all environments, locales, and situations. We saw one such example when we created the application for a number to words conversion. So, there is an enormous need for the development of new applications using R language.

The prudent way to solve problems using R language is to search for components of the required solution that are already available in R libraries and then building on top of that. This way the developer can create new applications in the least amount of time. Also, as the applications will use the tried and tested libraries, the applications are likely to be more robust.

We have come to the end of the section on the basics of R programming. This book discusses the essentials of R programming required for building our solution for emotion analysis and thus, is not meant to be a book on R programming.

In the next chapter, we will explore how to create web applications using Shiny programming.

### Points to remember

Conditions can be programmed in R using the if statement.

Loops can be programmed in R using and repeat statements.

The english() function in the english library provides functionality for converting numbers to words as per the European convention.

The wordcloud() function in the wordcloud library provides functionality for creating word clouds.

The **tm** library provides essential functions for Text Mining (however, there are other libraries in R which provide functions for Text Mining as well).

### Multiple choice questions

The syntax of a while statement is:

while(test\_expression) {Statement set}

while(val in sequence) {Statement set}

while(Statement block) {Statement set}

None of the above

The length function is used to get the:

Number of digits in a number

Number of elements in a vector

Number of characters in a string

None of the above

What is a recursive function?

When the function is called within the function itself

When the same function is called multiple times in a program

When one function is called in a for loop

None of the above

Which of the following statements is not valid?

Statements cannot be grouped together using braces '{' and '}'.'

Computation in R consists of sequentially evaluating statements.

Single statements are evaluated when a new line is typed at the end of the syntactically complete statement.

next is used to skip an iteration of a loop.

NLP stands for:

Near Linear Programming

Normal Length Programming

Natural Language Processing

None of these

The function used to create a Corpus is:

Corpus()

corpus()

Both a and b

None of these

We can create a Word Cloud using which of these functions?

WordCloud()

wordcloud()

Both a and b

None of these

The english() function from the english library in R can be used to convert a number to words as per which convention?

Any convention

Indian convention only

European convention only

Asian convention only

## Answers to MCQs

A

B

A

A

C

A

B

C

## Questions

Write the number to words conversion application for the European convention on your own. In the European convention, we have units, tens, hundreds, thousands, millions, billions, and trillions.

Write a number to words conversion application as per the convention used in your country. Also, generate the words in your native language like Hindi, Spanish, Thai, German, etc.

We discussed a program for creating a Word Cloud based on the frequency of the occurrence of the expense heads in the data. You would notice that in the Word Cloud, the expense heads with a higher frequency appear larger and the size of the expense heads reduces as their frequency reduces. Modify the program so that the size of the expense heads is determined by the amount spent under that expense head.

## Key terms

**Conditions:** Conditions are programming constructs where a computer program needs choosing between alternatives. As all constructs in computer programming are based on Boolean Algebra, the decisions from a condition construct in computer programming must result in binary decision – either TRUE or FALSE. In cases of more than 2 alternatives, multiple numbers of conditions must be combined to create the needed logic.

**Loops:** Loops are programming constructs where a computer program performs the same set of instructions for multiple number of data items. The loop programming construct has a controller which decides the number of times the loop would be executed. Generally, each pass of the loop over the set of instructions inside the loop are called

**Functions:** Functions are programming constructs where functional requirements, which may need to be executed multiple numbers of times in different situations from different points of one or more applications, are encapsulated in a single program unit. Functions are the basic building blocks of any application. Functions ensure code reuse and prevents code duplication.

## SECTION - 2

## CHAPTER 4

### Structure of a Shiny Application

In [Chapter 3: Developing Simple Applications in R](#), we developed two simple applications in R using all the usual programming constructs.

Once we have created an application, we need to make it available to the world. One of the ways to make the application available to the world is by creating a web application and publishing it on the Internet to be accessed by everyone. Creating a web application is possible by using the Shiny library of R. In this chapter, we will go through the structure of a Shiny application. In [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#) we will build two Shiny applications for number to words conversion and Word Cloud Generation.

## **Structure**

In this chapter, we will discuss the following topics:

Creating a Shiny application using RStudio

Structure of a Shiny program

## Objectives

After studying this unit, you should be able to:

Understand the structure and components of a Shiny application

Be able to create and execute the default Shiny application

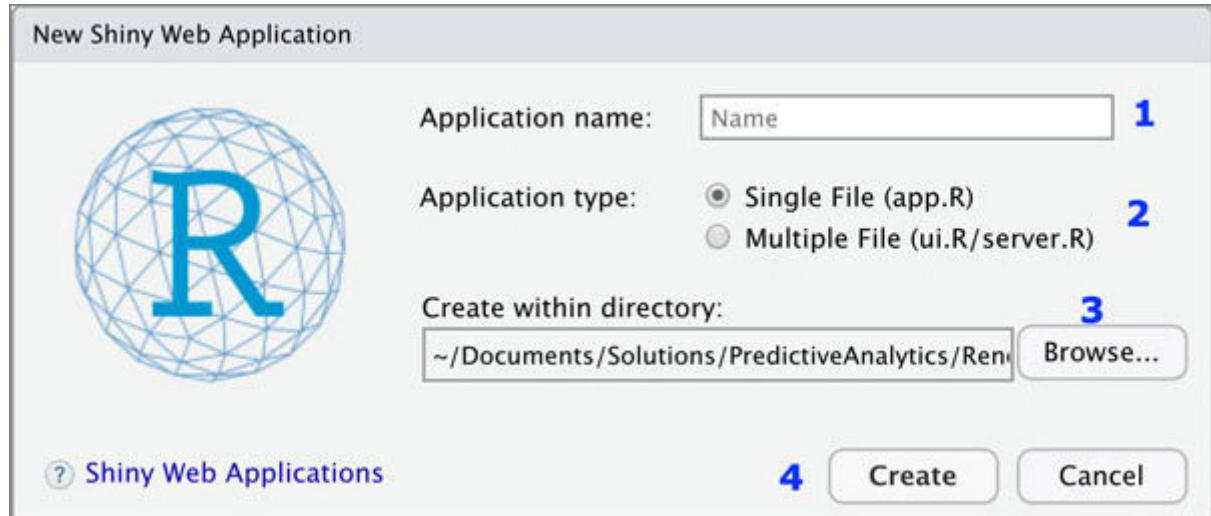
## [\*Creating a Shiny application using RStudio\*](#)

Shiny applications are based on client-server architecture. In other words, we have two components in the Shiny application – one component is the client program and the other component is the server program. In the client application, we program the **User Interface (UI)** which the users of the application see and interact with application (As Shiny applications are web applications, the UI is displayed in a browser). The server application is not visible to the users. The server application does all the processing required by the application and provides the results for the client application to display.

Shiny applications can be created using RStudio. In RStudio, click on the **Menu** options as follows to create a Shiny application:

**File | New File | Shiny Web App...**

The following window will pop up:



**Figure 4.1:** Create Shiny Application Dialog

Against application name (marked by 1), provide the name of the application. A directory is created by the name of the application where all the files for this application are stored. There are two options for creating files for the application. One option is to create both the client application and the server application in a single file and to name the file The other option is to create two separate files for the client application and the server application. The file containing the client application is named ui.R and the server application is named

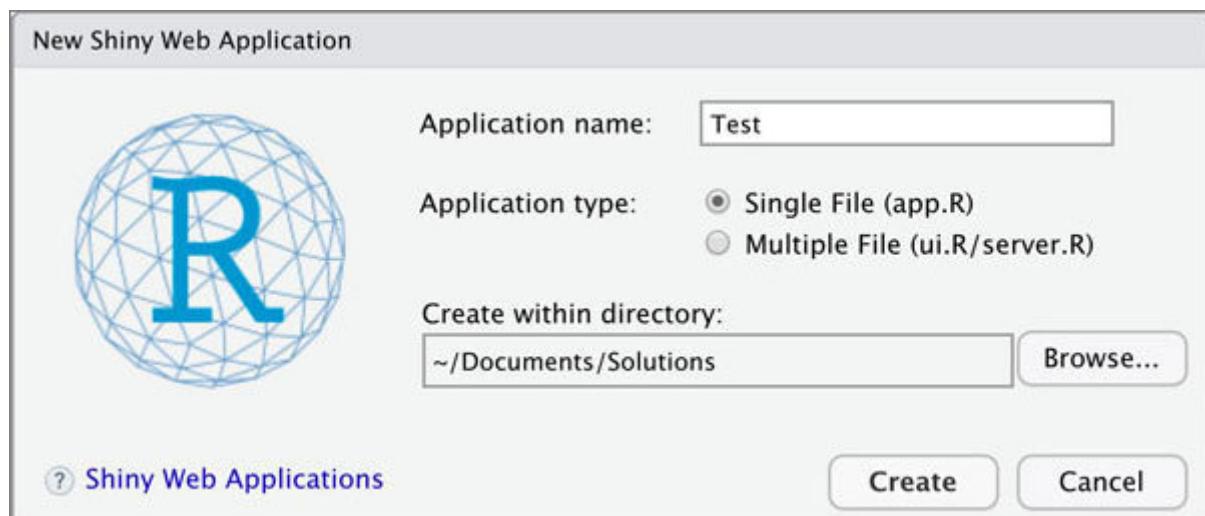
We can select how to create the Shiny application – single file or multiple files – by picking an option against the application type (marked by 2).

As discussed earlier, all files for a Shiny application need to be in a directory by the name of the application. We can choose where this directory would be created in our disk. We can browse our filesystem and select the directory under which the application

directory would be created by clicking the **Browse** button (marked by 3).

Once we have provided the above three information, we can click the Create button (marked by 4) to create the Shiny application. RStudio creates a default Shiny application. We can modify this default Shiny application as needed.

Let us create a Shiny application called Test. The dialog box should look as shown in the following screenshot:



**Figure 4.2: Create Shiny App Test**

On clicking the **Create** button, RStudio will create a file named app.R in the directory called Test under the directory you have specified (in the above example, this directory is

The contents of the file app.R should be as shown in the following code:

```
#  
# This is a Shiny web application. You can run the application by  
clicking  
# the 'Run App' button above.  
#  
# Find out more about building applications with Shiny here:  
#  
#     http://shiny.rstudio.com/  
#  
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) {
  <- renderPlot({
    # generate bins based on input$bins from ui.R
    x     <- 2]
    bins <- seq(min(x), max(x), length.out = input$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)

```

I make a few subtle changes to make the program more readable. The altered program is as shown below. The altered lines are marked by the comment in

```

#
# This is a Shiny web application. You can run the application by
clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#      http://shiny.rstudio.com/

```

```
#  
  
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(inputId = "v_bins",           ## ALTERED  
                  label = "Number of bins:", ## ALTERED  
                  min = 1,  
                  max = 50,  
                  value = 30)  
  
    ),  
    # Show a plot of the generated distribution  
    mainPanel(  
      plotOutput(outputId = "distPlot")        ## ALTERED  
    )  
  )  
  
  # Define server logic required to draw a histogram  
  server <- function(input, output) {  
    output$distPlot <- renderPlot({
```

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

# 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)
```

To understand the structure of a Shiny application, we will discuss the default Shiny application created by RStudio. In [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#) we will create specific Shiny applications for our need.

## Structure of a Shiny application

In the default code generated, you would have noticed that we need the library This is mandatory for every Shiny application. We may add any number of libraries to the Shiny application. However, we must add the shiny library.

As I have mentioned above, the Shiny application has two components – the client application and the server application. Let us discuss both the components separately.

## Client-side program

From the default code generated, I have extracted the code pertaining to the client-side program:

```
# 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(inputId = "v_bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    mainPanel(
      plotOutput(outputId = "distPlot")
    )
  )
)
```

Notice that the client-side program is a function named As I have mentioned earlier, the client-side program creates the UI to be displayed on a browser. The function `fluidPage()` creates a web page.

In the web page, we can use various kinds of components. All the components that are supported by HTML can be programmed using Shiny programming. However, we must note that the HTML components rendered will differ from browser to browser.

The basic structure of a web page is that we can create several panels inside the web page. Inside the panels, we can plant various types of controls. All the panels can be arranged in different Layouts. We will discuss various controls, panels and layouts in this book. Please note that this is not a book on Shiny *programming* and thus, we will only discuss Shiny Programming from the perspective of what we need for building our application for emotion analysis.

### Displaying title

In the above program, note that a titlePanel() has been used. We can pass a string as a parameter to the titlePanel() function and the string will appear as a title on the web page:

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

For this example, the title appears as shown below:

# Old Faithful Geyser Data

**Figure 4.3:** Application title

## The body of the web page

```
# Sidebar with a slider input for the number of bins
sidebarLayout(
  sidebarPanel(
    ...
  ),
  # Show a plot of the generated distribution
  mainPanel(
    ...
  )
)
```

Next, note that sidebarLayout() has been used in the program. sidebarLayout() provides two panels – sidebarPanel() and sidebarLayout() can be schematically thought of as shown in the following figure:



**Figure 4.4:** Conceptual view of the `sidebarLayout()`

So, if we put together all the panels used in the program (that is title panel, sidebar panel, and main panel), we should expect a web page as shown in the following figure:



**Figure 4.5:** Conceptual view of the web page created by the default *Shiny Application*



## Taking input

We notice that in the the program has one control. The control used is a slider input control and it is created using the `sliderInput()` function. Slider input control is used to accept an input from the user. This program intends to take an input from the user through the slider input control:

```
sliderInput(inputId = "v_bins",
label = "Number of bins:",
min = 1,
max = 50,
value = 30)
```

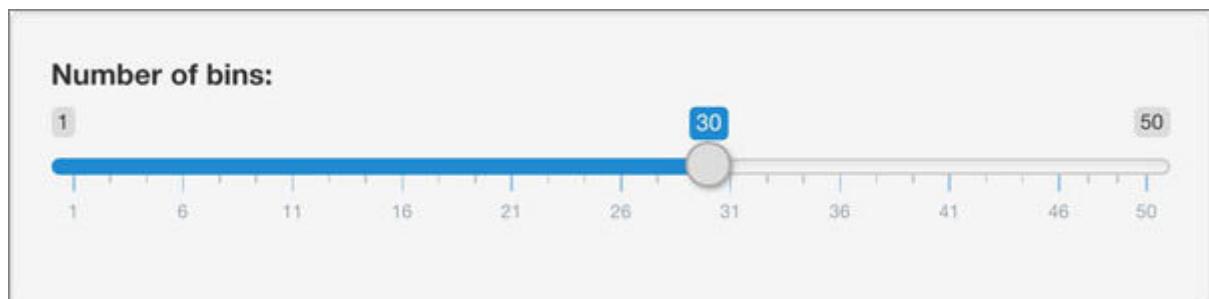
We will discuss several controls provided by the **shiny library** in this book. However, I repeat that this is not a book on Shiny programming and thus, the list of controls discussed in the book is not exhaustive. However, having understood the controls used in this book, it should be possible to program any of the controls not discussed in this book.

Notice that the first parameter to the `sliderInput()` function is This parameter associates a variable to the control. The variable we have associated is Using the variable we can refer to the value represented by this slider input control. Later, when we discuss the server-side program, we will see how to use this value.

The remaining parameters of `sliderInput()` function are as required for creating the slider input. The `label` parameter states what text will be displayed where the slider input is planted. Through the provided label, we can make the user aware of the input that is being requested for. For example, this slider input is used to capture the number of bins that the histogram needs to display.

The `min` parameter determines the minimum value that can be accepted from this slider input control. Similarly, the `max` parameter determines the maximum value that can be accepted from this slider input control. The `value` parameter determines what would be the default value of the slider control when the web page is displayed for the first time.

When this slider input is first displayed on the web page, it would appear as shown in the following screenshot:



**Figure 4.6:** Slider Input initial display

## Displaying output

In this program, a histogram is displayed in the main panel. The histogram is generated by the server-side of the Shiny application. So, we need to make a connection between the client-side and the server-side programs to be able to display the histogram.

Let us discuss the code for the same at the client-side program first:

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

The main panel contains a call to the `plotOutput()` function. `plotOutput()` is one of the many output functions in the shiny library. The **library `plotOutput()`** function is used to display a plot. Similarly, there are other output functions like etc. which are used for other types of output.

This book discusses quite a few output functions provided for Shiny programming. However, we will not discuss the exhaustive list of output functions as this book is not focused on Shiny programming.

Notice that we have passed the parameter **outputId** to the **plotOutput()** function. This parameter defines a variable (in this case, the variable is named `Using`) using this variable, we can connect the client-side program with the server-side program. We will discuss this shortly when studying the server-side code.

## Server-side program

At the server side, we program everything that needs computations of any kind. The computation could be validations, calculations, business process implementation, etc. This is same as what any server-side program does in any client-server application.

Let us examine the code at the server side:

```
# 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$v_bins + 1) # ALTERED

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

The server-side program is contained in a function named In the above program, the **server()** function takes two parameters – input and output. However, the **server()** function can also take a third

parameter - session. We will discuss this in [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#)

This server-side program generates a histogram for data available in R libraries called faithful. We read the data in the second column of the faithful data frame into a variable Then we generate a sequence using the **seq()** function. The sequence is generated between the minimum and the maximum values of the data in the variable The intervals are controlled by the value in the variable

Remember that the variable v\_bins was defined in the client-side program to capture the value of the slider. Notice that Input is the parameter that we passed to the **server()** function. It contains all the variables passed as input. So, as is a variable defined in the client-side program to capture an input, it is available to the server-side program through the input variable.

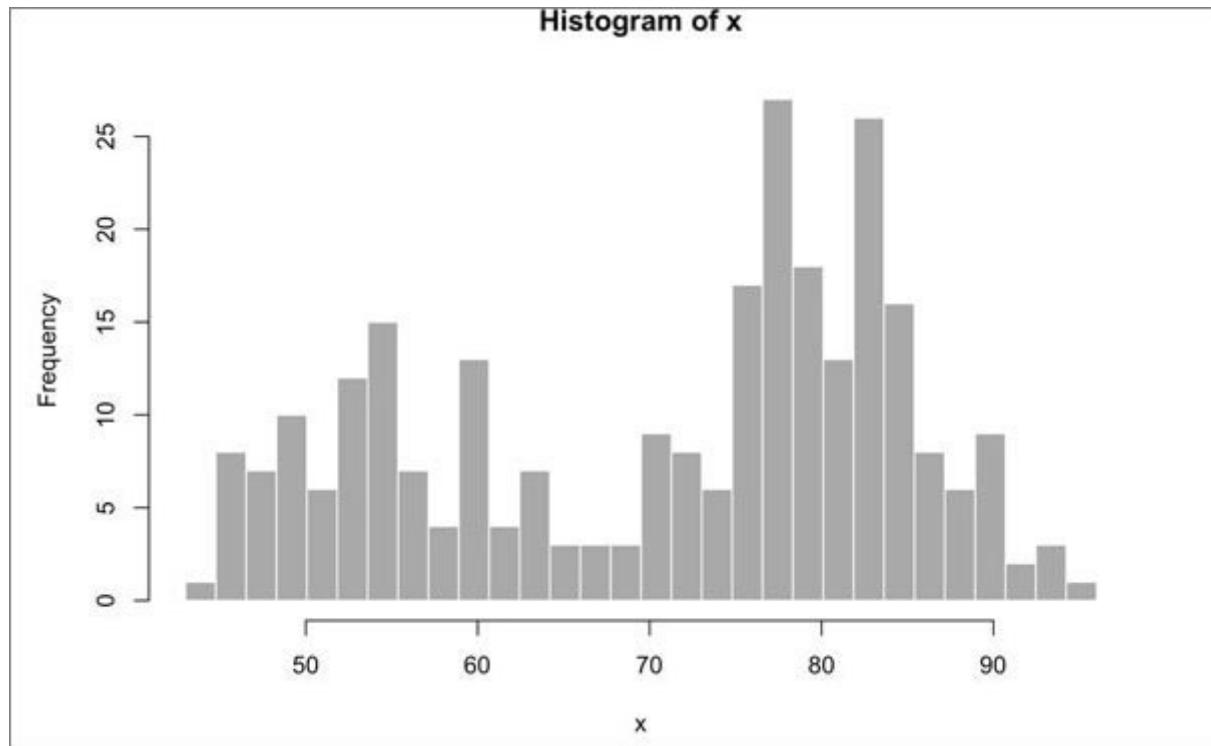
Next, notice the use of the **renderPlot()** function. The **renderPlot()** function is used to render a plot. This function works hand-in-hand with the **plotOutput()** function. When we use the **renderPlot()** function in the server-side program, we need to use the **plotOutput()** function in the client-side program.

Similarly, if we use the **renderText()** function in the server-side program, we need to use the **plotText()** function in the client-side program. We will discuss several render functions in this book. However, we will not discuss the exhaustive list of render functions in this book as this book is not focused on Shiny programming.

Notice that we defined the function `output$distPlot()` on the server side. Remember that we defined the variable `distPlot` in the client-side program as an output variable. So, this variable is available with the parameter `output` to the `server()` function. We can reference this variable using `The render` function pack the output in the `output` variables and they become available to the client-side programs.

In this case, the server-side program has only one output function to generate the histogram based on the number of bins provided as input. You will notice in [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application 2](#) that we will require multiple output functions for to create a web application.

The default output from this server-side code is as shown in the following screenshot:



**Figure 4.7:** Default output from the default Shiny Application

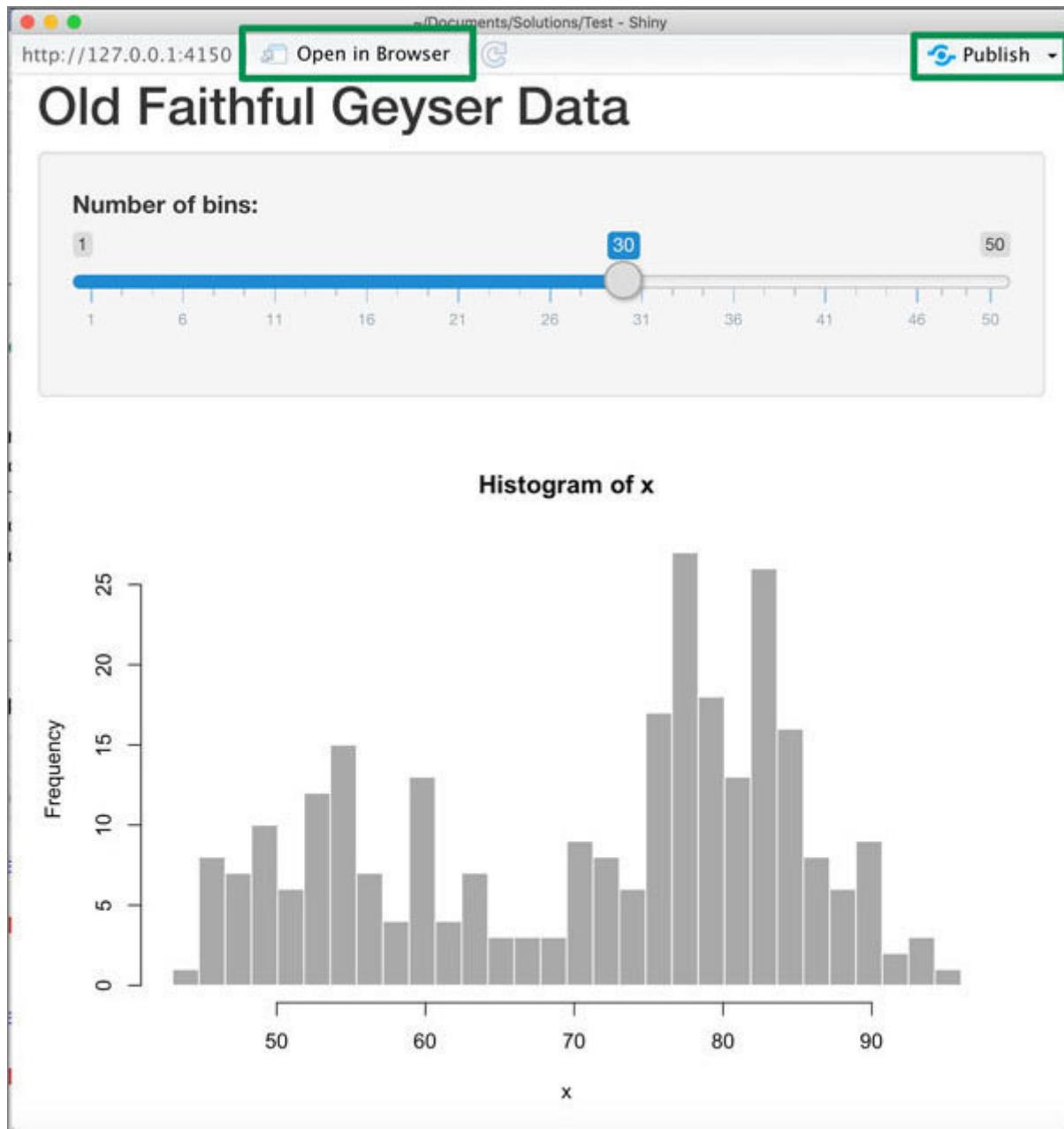
## [Running the application](#)

The code to run the Shiny application is provided in the following code:

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

The function `shinyApp()` runs the Shiny application. Notice that the `shinyApp()` function takes two parameters and these are the names of the client-side and the server-side program functions.

The output generated when this default program is run is as shown in the following screenshot:



**Figure 4.8:** Output when the default Shiny Application is run

The above output is from the simulator provided by RStudio. Using the button Open in browser, the application can be opened in any browser. When the application is opened in a browser, it will run locally from the localhost. In order to make the application available on the World Wide Web, the application needs to be published on a server. Notice the **Publish** button on

the right. Clicking this provides options to publish the application to a server. We will discuss this aspect in 5: *Shiny Application*

### *Creating separate files for client-side and server-side programs*

If we choose to keep the client-side and the server-side programs in separate files, we need to choose the multiple-file option when creating a Shiny application using RStudio. In this case, two files - ui.R and – are created in the application as follows.

[ui.R](#)

```
library(shiny)
```

```
# Define UI for application that draws a histogram  
shinyUI(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")
```

```
  )
```

```
  )
```

```
))
```

```
server.R
```

```
#  
# This is the server logic of a Shiny web application. You can run  
the  
# application by clicking 'Run App' above.
```

```
#  
# Find out more about building applications with Shiny here:  
#  
#     http://shiny.rstudio.com/  
#
```

```
library(shiny)
```

```
# Define server logic required to draw a histogram  
shinyServer(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')  
    })  
})
```

## Conclusion

This chapter introduces the learner to Shiny programming. Using Shiny programming, it is possible to create almost all kinds of interactive and static web applications. Shiny applications follow client-server architecture. The client programs create the user interface and the server programs conduct the needed processing.

In [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#) we will discuss the creation of two Shiny applications for specific requirements. After the discussion on emotion analysis, we will learn how to implement those algorithms into Shiny applications.

### Points to remember

Web applications can be created using Shiny programming.

shiny library is essential for creating Shiny applications.

There are two options for creating Shiny applications – single file option and multiple files option.

### Multiple choice questions

By default, the name of the function on the client-side is:

client()

ui()

UI()

None of these

By default, the name of the function on the server-side is:

server()

service()

Server()

None of these

The function used to run Shiny Applications is:

runShinyApp()

`invokeShinyApplication()`

`shinyApp()`

None of these

The function used to create the Slider Input is:

`createSlider()`

`sliderInput()`

`slider()`

None of these

## Answers to MCQs

B

A

C

B

## Questions

Take any data from your office or any academic field. Alter the Shiny Application discussed application in this chapter to create a pie chart that displays one of the attributes of this data in a Shiny application.

## Key terms

**Client-Server server architecture:** Client-Server server architecture is a computing model in which the server hosts, delivers, and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection.

## CHAPTER 5

### Shiny Application 1

In [Chapter 3: Developing Simple Applications in R](#), we discussed the application for converting number to words. And in [Chapter 4: Structure of Shiny Application](#), we discussed how to create a Shiny application. So far, we have laid the foundation for what we will discuss in this chapter.

In this chapter, we will create the first Shiny application using the knowledge discussed in this book so far. We discussed the program for number to words convertor in [chapter 3: Developing Simple Applications in](#). However, we could use that program using a **Character User Interface (CUI)** from the Command Line. Now, we will create a web application through which we could get the functionality for number to words conversion. We already know from [Chapter 4: Structure of Shiny Application](#) that Shiny applications provide a **Graphical User Interface (GUI)**.

## **Structure**

In this chapter, we will discuss the following topics:

Designing the user interface

Coding the user interface

Coding the server-side code

Publishing Shiny application to the World Wide Web

## Objectives

After studying this unit, you should be able to:

Understand how to create Shiny applications

Publish Shiny applications to the World Wide Web

## Designing the user interface

Before we start creating the Shiny program, we need to plan the UI the users using this application. To understand the designing aspect, I will present the application that I have created and work backwards. The way I have made the application is one of a million ways the same could have been done. I would encourage you to understand the concepts and use your creativity to make application which is much more attractive.

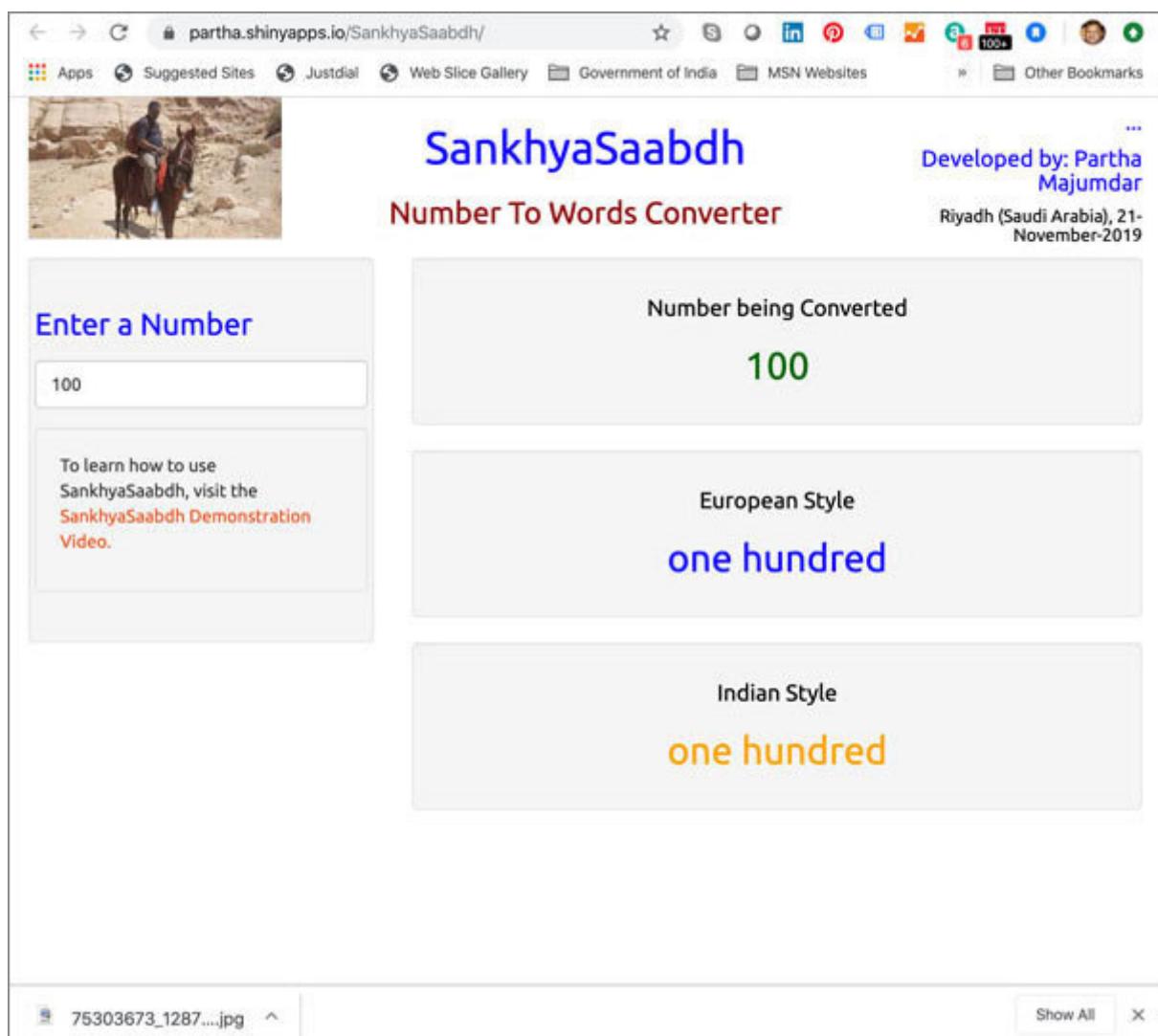
I named my application for number to words conversion as SankhyaSaabdh. As a practice, I would encourage you to get into the habit of naming your applications suitably.

While travelling by train from Howrah to Delhi somewhere in the 1990s, one Marwadi Businessman explained to me the importance for creating an identity for any product that one develops. When we develop any product, it always starts from a very small scale. However, we should always start with a grand vision. Any big success story always starts with a very small initiative.

As an illustration, you would have realized that the application for number to words conversion is a very simple program a computer programmer could write. However, now, having included this program in a book, it has a much bigger dimension.

## The UI for SankhyaSaabdh

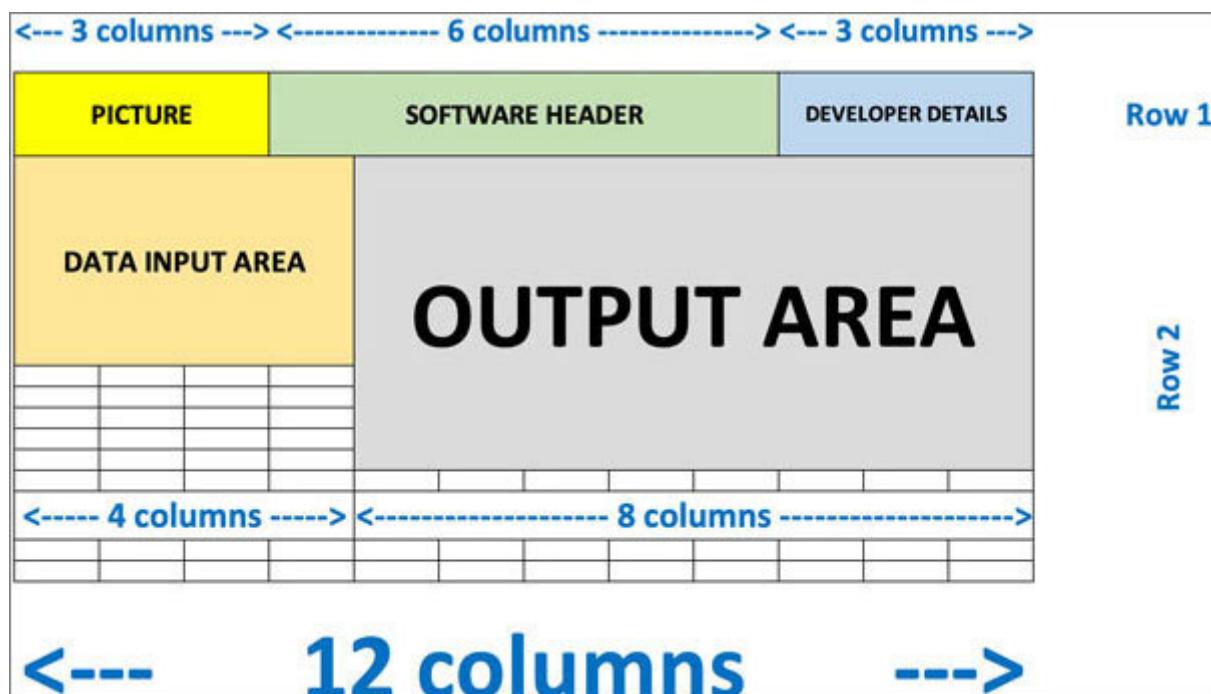
Provided below is the UI I created for the number to words conversion application **SankhyaSaabdh**.



**Figure 5.1:** User Interface for SankhyaSaabdh

This application is available at the URL  
<https://partha.shinyapps.io/SankhyaSaabdh/>

Let us break down the UI into its components. Given below is the schematic diagram of how the GUI for SankhyaSaabdh is planned:



**Figure 5.2:** Schematic plan of the SankhyaSaabdh UI

The first aspect to note is that the browser window is divided into 12 columns. So, we need to plan the user interface by using these 12 columns. I have planned the user interface for SankhyaSaabdh in 2 rows. The first row is the header for the application and the second row is the body for the application.

In the first row, I have chosen to keep 3 sections. The 3 sections have 3 columns, 6 columns and 3 columns as shown in [figure](#) In

the left section of the 3 columns on the header row, I want to plant a picture. In the middle section of the 6 columns of the header row, I want to display some text which will state the application name and its purpose. In the right section of the 3 columns of the header row, I want to provide some text stating the developer details.

In the second row, I have chosen to keep 2 sections of 4 columns and 8 columns. In the left-hand section of the 4 columns on the body row, I want to take user input. On the right-hand section of the 8 columns on the body row, I want to display the output.

## Coding the user interface

Now that we have planned our user interface, let us discuss the code required to create this user interface. We need to discuss the following at this stage:

How to display *images* in a Shiny application?

How to display *text* in a Shiny application?

How to take *number only input* in a Shiny application?

How to display *text output* in a Shiny application?

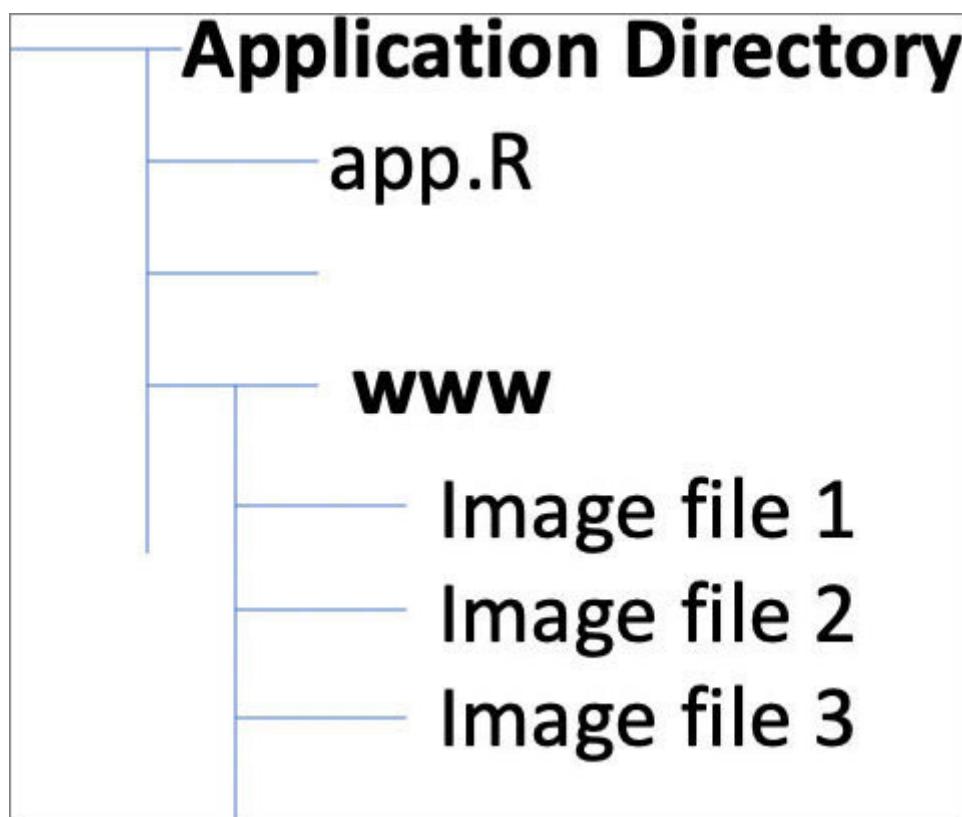
How to create the

We will learn more about the different aspects of user interface programming for Shiny application in [Chapter 6: Shiny Application](#)

## Displaying images in Shiny applications

To display images in Shiny applications, we need to create a directory named **www** under the **Application** directory. All the images to be displayed by the Shiny application must be stored in this directory. Images cannot be stored in subdirectories under the directory

So, the generic directory for Shiny Applications with respect to displaying images is as shown in the following screenshot:



**Figure 5.3:** Generic directory structure of Shiny Applications w.r.t. images

Specifically, in SankhyaSaabdh, we use only 1 image and so it needs to be stored as shown in the following screenshot:



**Figure 5.4:** *SankhyaSaabdh directory structure*

To display the image, we need the `img()` function as shown in the following code:

```
img(src="ParthalnPetra.jpg", width=200, height=112)
```

The name of the image file needs to be provided as an input to the parameter `src` of the `img()` function. The `width` and the `height` parameters of the `img()` function can be used to set the dimension of the image to be displayed.

So, we have created the following portion of the user interface for SankhyaSaabdh:



**Figure 5.5:** Image display for SankhyaSaabdh

## Displaying text in Shiny applications

Text can be displayed in Shiny application using functions parallel to all the HTML tags. I will discuss some of the header functions to display text in the SankhyaSaabdh application. In the middle section of the **header** row, we would like to display the application name and the application purpose. This can be achieved using the following code:

```
h1("SankhyaSaabdh", style="color:blue; text-align: center;"),  
h3("Number To Words Converter", style="color:darkred; text-align:  
center;")
```

In the above code, I have used the functions `h1()` and `h3()`. These functions are equivalent to

and

tags in HTML. Similarly, we have **h2()**, **h4()**, **h5()**, and **h6()** functions.

To these functions, we need to pass the text that needs to be displayed in the respective header. Also, we can provide a style parameter. In the style parameter, we can provide any styles supported by HTML. In the above code, you can see that I have set the color of the text and set the alignment of the text.

Similarly, in the right section of the header row, we would provide the details of the developer and the application development. This is accomplished with the following code:

```
h4("...", style="color:blue; text-align: right;"),  
h4("Developed by: Partha Majumdar", style="color:blue; text-align:  
right;"),  
h5("Riyadh (Saudi Arabia), 21-November-2019", style="color:black;  
text-align: right;")
```



**Figure 5.6:** SankhyaSaabdh header

## Accepting number only input in Shiny applications

We need to allow the users to be able to enter a number which SankhyaSaabdh will convert to words. We need to ensure that the user can only input a number and the input should accept only the keystrokes of the number keys.

This can be accomplished using the following code:

```
numericInput("v_input",
h3("Enter a Number", style="color:blue; text-align: center;"),
value = "100")
```

The function `numericInput()` only accepts numeric input. The first parameter is the variable associated with the control; in this case, it is `v_input`. The second parameter is the label that needs to be displayed in front of the input area. Notice that I have passed an HTML header `h3` as the label. The third parameter – `value` – sets the default value for the control. To distinguish this input area from the rest of the screen, we can enclose the `numericInput()` function within a `div`. The output created is as shown in the following screenshot:

# Enter a Number

100

To learn how to use  
SankhyaSaabdh, visit the  
[SankhyaSaabdh Demonstration  
Video.](#)

**Figure 5.7:** Numeric input embedded in a well panel

Notice that we have a panel inside the well panel. In the smaller panel, we have placed a hyperlink. Provided below is the complete code for creating this section of SankhyaSaabdh:

```
wellPanel(  
  fluidRow(  
    numericInput("v_input",
```

```
h3("Enter a Number", style="color:blue; text-align: center;"),  
  
value = "100")  
,  
fluidRow(  
wellPanel(  
p("To learn how to use SankhyaSaabdh, visit the ",  
a("SankhyaSaabdh Demonstration Video.",  
href = "https://youtu.be/Gbhy1atxybo",  
target="_blank")  
)  
)  
)  
)
```

Notice the use of the **a()** function for creating a hyperlink. The parameter **target="\_blank"** ensures that the hyperlink is opened in a new tab. Also, notice the use of the **p()** function for creating a paragraph. The **p()** function is the equivalent to the HTML tag.

We will discuss `fluidRow()` shortly in the section *Creating layouts in Shiny applications*.

## Displaying the text output in Shiny applications

Once the user inputs the number to convert to words, SankhyaSaabdh should convert the number and generate the text for the number expressed in words. This text needs to be displayed to the user. So, we need to output the text. Remember that in [Chapter 4: Structure of Shiny](#) we discussed how to output plots in a Shiny Application.

To display the text output, we can use the following code:

```
wellPanel(  
  h4("Number being Converted", style="color:black; text-align:  
  center;"),  
  h2(textOutput("rawNumber"), style="color:darkgreen; text-align:  
  center;")  
,  
  wellPanel(  
    h4("European Style", style="color:black; text-align: center;"),  
    h2(textOutput("europeanStyle"), style="color:blue; text-align: center;")  
,  
    wellPanel(  
      h4("Indian Style", style="color:black; text-align: center;"),  
      h2(textOutput("indianStyle"), style="color:orange; text-align: center;")  
)
```

The function `textOutput()` is used to display the text output. Just like the `plotOutput()` function to display plots (discussed in [Chapter 4: Structure of Shiny](#)) the `textOutput()` function takes an output variable as a parameter. The variables in the above code – and `indianStyle` – are the output variables. We will make use of them in the server-side code to connect to the client-side code.

Also, notice that within the `wellPanel()` functions, we have provided two components each. In each panel, the first component displays a static text and the second component displays the text output. We have separated text output. The code for these components by a comma. Inside a panel, we can plant any number of components. The code for each component should be separated by a comma.

**Putting a comma after the last component within a panel will cause the Shiny Application to not work. No compilation error will be generated. However, the application will not work.**

The output of this code is as shown in the following screenshot:

Number being Converted

100

European Style

one hundred

Indian Style

one hundred

*Figure 5.8: Output from SankhyaSaabdh*

## [Creating layouts in Shiny applications](#)

There are several sets of functions for creating layouts in Shiny Applications. I will discuss about the *fluid page layout*. Using the fluid page layout, almost any layout can be created in Shiny applications.

Following code is for creating the first row of our UI:

```
# Define UI for application
ui <- fluidPage(
  # Application title
  fluidRow(
    column(width = 3,
           ),
    column(width = 6,
           ),
    column(width = 3,
           )
  ),
  )
```

A fluid page layout consists of rows which in turn include columns. Rows exist for the purpose of making sure that their elements appear on the same line (if the browser has adequate width). Columns exist for the purpose of defining how much horizontal space within a 12-unit wide grid should its elements occupy. fluid pages scale their components in real-time to fill all the available browser width.

The function to create a fluid page layout is `fluidPage()`. The function to create rows inside the fluid page is `fluidRow()`. We can create more than one Fluid Row inside a fluid page. Calls to each `fluidRow()` should be followed by a comma, except for the call to the last `fluidRow()` function within a `fluidPage()` function.

To create columns inside a fluid row, we use the function `column()`. The `column()` function takes 2 parameters. The first parameter is the width of the column. The second parameter is the component to be placed inside the column. We can place one or more components inside a column. Each component code should be followed by a comma, except for the code of the last component in the column.

Following is the outline code of the second row of SankhyaSaabdh:

```
fluidRow(  
  column(width = 4,  
    wellPanel(  
      fluidRow(  
    ),
```

```
fluidRow(  
  wellPanel(  
    )  
    )  
    )  
  ),  
  column(width = 8,  
  fluidRow(  
    column(width = 12,  
    wellPanel(  
  
  ),  
  wellPanel(  
    ),  
  wellPanel(  
    )  
    )  
    )  
    )  
  )
```

**Notice (in the code above) that inside every column, the space is divided into 12 columns.**

## Coding the server side of SankhyaSaabdh

In [Chapter 3: Developing Simple Applications in](#) we discussed the code for the conversion of numbers to words. The function we wrote was named We make use of the same function. In SankhyaSaabdh, we will declare this function globally so that it is available throughout the scope of SankhyaSaabdh.

The service-side code is provided in the following code:

```
# Define server logic required
server <- function(input, output, session) {

  output$rawNumber <- renderText({
    req(input$v_input)
    return(format(round(input$v_input, 0), nsmall = 0, big.mark = ","))
  })

  output$europeanStyle <- renderText({
    req(input$v_input)
    return(
      ifelse(input$v_input <= v_largest_number,
        english::english(round(input$v_input,0)),
        "Number too large"
      )
    )
  })
}
```

```

output$indianStyle <- renderText({
  req(input$v_input)
  return(
    ifelse(input$v_input <= v_largest_number,
    numbers2wordsIndia(round(input$v_input,0)),
    "Number too large"
  )
)
})

```

Notice that we have three output functions in the server-side code. The two output functions are associated with the three output variables – and `indianStyle` – declared in the client-side code. All these variables need to output text and so we have called the `renderText()` function to return the output.

Notice the code The `req()` function ensures that the associated variable definitely has a value in it. So, we proceed with the rest of the function only if the variable `v_input` has a valid value.

Notice the use of the variable This variable contains a sentinel value. We can set this variable to a large value according to the configuration of the server where this program is deployed. This is to ensure that the number to words conversion function does not face a situation where a very large number is provided as input to the function. `v_largest_number` variable is set as a global variable in the program.

## The complete code of SankhyaSaabdh

Following is the complete code of SankhyaSaabdh:

```
##### LOAD THE REQUIRED PACKAGES
```

```
#####
```

```
library(shiny)
library(shinythemes)
library(english)
```

```
##### CONVERT NUMBER TO WORDS IN INDIAN
STYLE #####
```

```
numbers2wordsIndia <- function(x){
  helper <- function(x){
    if (x < 0) {return(paste(x, "is negative!"))}

  digits <- rev(strsplit(as.character(x), "")[[1]])
  nDigits <- length(digits)
  if (nDigits == 1) as.vector(ones[digits])
  else if (nDigits == 2)
    convert2DigitNumbers(x)
  else if (nDigits == 3)
    convert3DigitNumbers(x)
  else if (nDigits == 4 || nDigits == 5)
    convertThousands(x)}
```

```
else if (nDigits == 6 || nDigits == 7)
convertLakhs(x)
else if (nDigits == 8 || nDigits == 9)
convertCrores(x)
else

trim(paste(numbers2wordsIndia(floor(x/10000000)), "crore",
convertCrores(x %% 10000000)))
}
```

```
convert2DigitNumbers <- function(x) {
if (x > 0 && x <= 99) {
digits2DigitNumber <- rev(strsplit(as.character(x), "")[[1]])
if (x <= 9) as.vector(ones[digits2DigitNumber])
else if (x <= 19) as.vector(teens[digits2DigitNumber[1]])
else {
if (digits2DigitNumber[1][1] == "o") {
trim(paste(tens[digits2DigitNumber[2]]))
}
else {
trim(paste(tens[digits2DigitNumber[2]],
as.vector(ones[digits2DigitNumber[1]])))
}
}
}
}
}
}
```

```
convert3DigitNumbers <- function(x) {
if (x > 0 && x <= 999) {
if (x < 100) convert2DigitNumbers(x)
else {
```

```
digits3DigitNumber <- rev(strsplit(as.character(x), ""))[[1]]  
trim(paste(ones[digits3DigitNumber[3]], "hundred and",  
convert2DigitNumbers(makeNumber(digits3DigitNumber[2:1]))))  
}  
}
```

```
}
```

```
convertThousands <- function(x) {  
if (x > 0 && x <= 99999) {  
if (x < 1000) convert3DigitNumbers(x)  
else {  
digitsThousands <- rev(strsplit(as.character(x), ""))[[1]]  
if (x <= 9999)  
trim(paste(ones[digitsThousands[4]], "thousand",  
convert3DigitNumbers(makeNumber(digitsThousands[3:1]))))  
else  
trim(paste(convert2DigitNumbers(makeNumber(digitsThousands[5:4])),  
, "thousand",  
convert3DigitNumbers(makeNumber(digitsThousands[3:1]))))  
}  
}  
}
```

```
convertLakhs <- function(x) {  
if (x > 0 && x <= 9999999) {  
if (x < 100000) convertThousands(x)  
else {  
digitsLakhs <- rev(strsplit(as.character(x), ""))[[1]]  
if (x <= 999999)  
trim(paste(ones[digitsLakhs[6]], "lakh",
```

```
convertThousands(makeNumber(digitsLakhs[5:1]))))  
else  
  
trim(paste(convert2DigitNumbers(makeNumber(digitsLakhs[7:6])),  
"lakh",  
convertThousands(makeNumber(digitsLakhs[5:1]))))  
}  
}  
}
```

```
convertCrores <- function(x) {  
if (x > 0 && x <= 999999999) {  
if (x < 10000000) convertLakhs(x)  
else {  
digitsCrores <- rev(strsplit(as.character(x), "")[[1]])  
if (x <= 9999999)  
trim(paste(ones[digitsCrores[8]], "crore",  
convertLakhs(makeNumber(digitsCrores[7:1]))))  
else  
trim(paste(convert2DigitNumbers(makeNumber(digitsCrores[9:8])),  
"crore",  
convertLakhs(makeNumber(digitsCrores[7:1]))))  
}  
}  
}
```

```
trim <- function(text){  
#Tidy leading/trailing whitespace, space before comma  
text=gsub("^\\ ", "", gsub("\\ *$", "", gsub("\,",",",text)))  
#Clear any trailing " and"  
gsub(" and$","",text)
```

```
}
```

```
makeNumber <- function(...) as.numeric(paste(..., collapse=""))
```

```
#Disable scientific notation
```

```
opts <- options(scipen=100)
```

```
on.exit(options(opts))
```

```
ones <- c("zero", "one", "two", "three", "four", "five", "six", "seven",  
"eight", "nine")
```

```
names(ones) <- 0:9
```

```
teens <- c("ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen",  
"sixteen", "seventeen", "eighteen", "nineteen")
```

```
names(teens) <- 0:9
```

```
tens <- c("twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty",  
"ninety")
```

```
names(tens) <- 2:9
```

```
x <- round(x)
```

```
if (length(x) > 1 && x >= 0) return(trim(sapply(x, helper)))
```

```
helper(x)
```

```
}
```

```
##### GLOBAL VARIABLES #####
```

```
v_largest_number <- 999999999999999
```

```
# Define UI for application
```

```
ui <- fluidPage(
```

```
theme = shinytheme("united"),  
  
# Application title  
fluidRow(  
  
  column(width = 3, img(src="ParthaInPetra.jpg", width=200,  
height=112)  
,  
  column(width = 6,  
h1("SankhyaSaabdh", style="color:blue; text-align: center;"),  
h3("Number To Words Converter", style="color:darkred; text-align:  
center;")  
,  
  column(width = 3,  
h4("...", style="color:blue; text-align: right;"),  
h4("Developed by: Partha Majumdar", style="color:blue; text-align:  
right;"),  
h5("Riyadh (Saudi Arabia), 21-November-2019", style="color:black;  
text-align: right;")  
)  
,  
  
  fluidRow(  
    column(width = 4,  
    wellPanel(  
      fluidRow(  
        numericInput("v_input",  
        h3("Enter a Number", style="color:blue; text-align: center;"),  
        value = "100")  
,  
        fluidRow(  
          ))
```

```
wellPanel(  
  p("To learn how to use SankhyaSaabdh, visit the ",  
  
  a("SankhyaSaabdh Demonstration Video.",  
    href = "https://youtu.be/Gbhy1atxybo",  
    target="_blank")  
)  
)  
)  
)  
,  
column(width = 8,  
fluidRow(  
  column(width = 12,  
  wellPanel(  
    h4("Number being Converted", style="color:black; text-align:  
      center;"),  
    h2(textOutput("rawNumber"), style="color:darkgreen; text-align:  
      center;")  
,  
    wellPanel(  
      h4("European Style", style="color:black; text-align: center;"),  
      h2(textOutput("europeanStyle"), style="color:blue; text-align: center;")  
,  
      wellPanel(  
        h4("Indian Style", style="color:black; text-align: center;"),  
        h2(textOutput("indianStyle"), style="color: orange; text-align:  
          center;")  
)  
)  
)
```

```
)  
)  
)  
  
# Define server logic required  
server <- function(input, output, session) {  
  output$rawNumber <- renderText({  
    req(input$v_input)  
    return(format(round(input$v_input, o), nsmall = o, big.mark = ","))  
  })  
  output$europeanStyle <- renderText({  
    req(input$v_input)  
    return(  
      ifelse(input$v_input <= v_largest_number,  
            english::english(round(input$v_input,o)),  
            "Number too large"  
      )  
    )  
  })  
  
  output$indianStyle <- renderText({  
    req(input$v_input)  
    return(  
      ifelse(input$v_input <= v_largest_number,  
            numbers2wordsIndia(round(input$v_input,o)),  
            "Number too large"  
      )  
    )  
  })  
}  
}
```

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

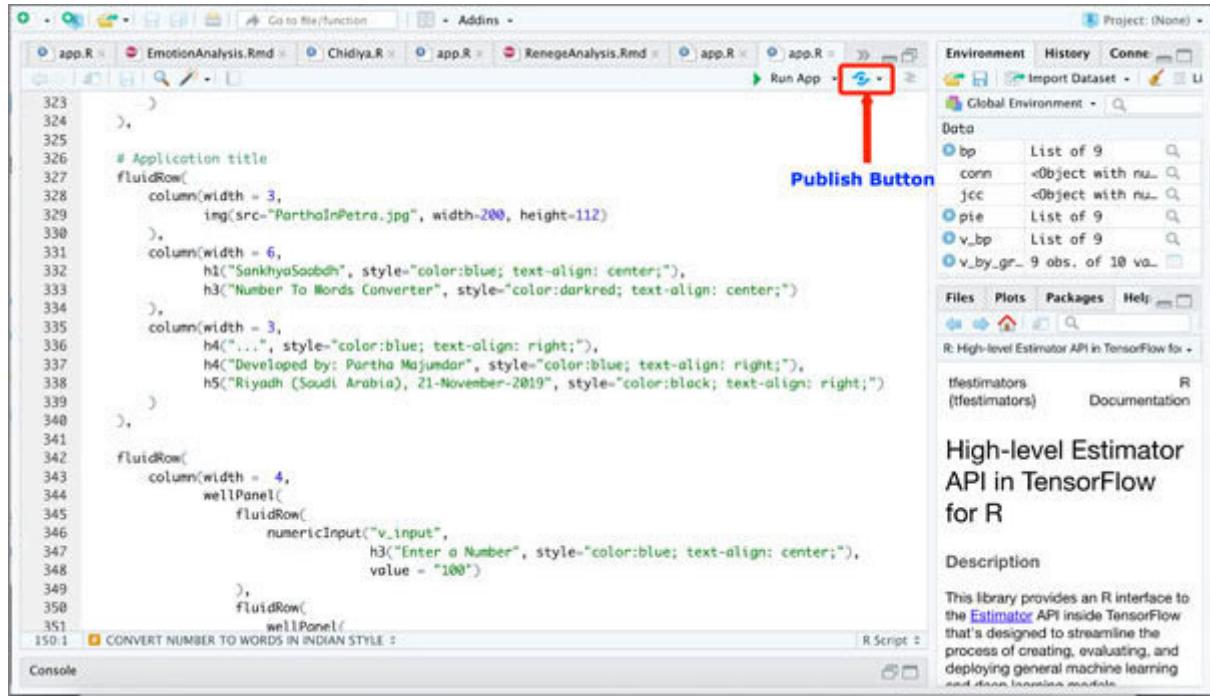
## Publishing a Shiny application

Once we have developed our Shiny application, we need to publish it to the World Wide Web so that it is assessable for everyone to use around the world. The simplest way to get a Shiny application published is to deploy the Shiny application to the Shiny server.

To access the Shiny server, one needs to create an account on the Shiny server. To create an account on Shiny server, visit <https://www.shinyapps.io/>

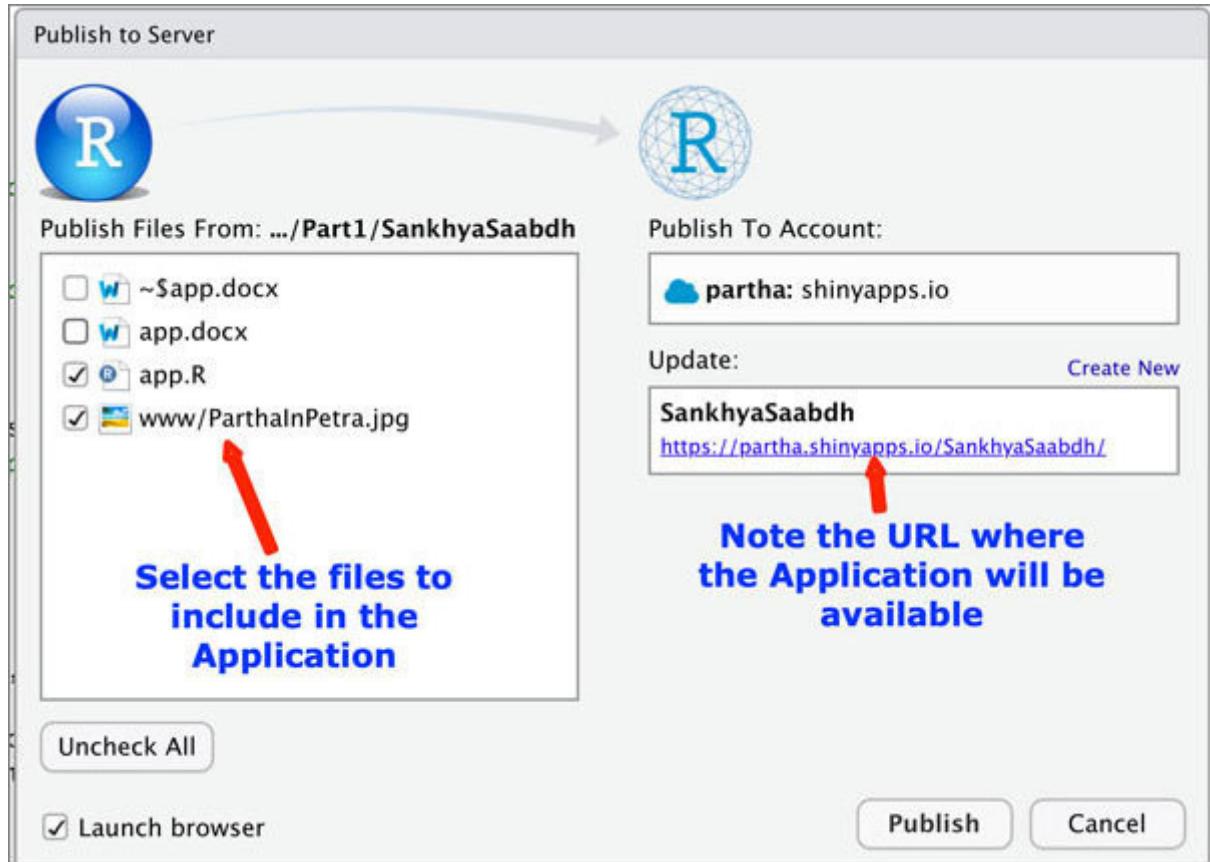
There are both free and paid options available for membership on Shiny server. Once the Shiny account is set up, we can use RStudio to publish the developed Shiny application on the Shiny server. The process is described as follows.

In RStudio, notice the Publish button as shown in the following screenshot:



**Figure 5.9:** ShinyApp Publish button for Shiny App in RStudio

When you click this button, two options should be provided. One option is to publish on RPubs and the other option is to publish on Shiny server. **RPubs** is used to publish documents. So, select the Shiny Server option as have to publish an application. On selecting the option, we will get an interface as shown in the following screenshot:



**Figure 5.10:** Shiny App publish window

In the above window, on the left-hand panel, select the files that need to be included in the Shiny application. Take note of the Shiny account where the Shiny application will be published. And also take note of the URL where the Shiny application will be available. Having ascertained all of the above, click the Publish button to publish the Shiny application.

## Conclusion

In this chapter, we created a slightly more complex Shiny application. We used the fluid page layout and the components of the same. We discussed how to display images, accept numeric input, and make multiple outputs in a Shiny application. We also made use of global functions and global functions and global variables.

In the next chapter, we will create a far more complex Shiny Application and learn application how to make use of JavaScript in Shiny applications.

### Points to remember

The browser page has 12 columns to frame the output.

There are 12 columns in every frame created inside a browser page.

The **fluidPage()** function can be used to create a fluid page layout.

The **fluidRow()** function can be used to create rows inside a fluid page.

The **column()** function can be used to create columns inside a fluid row.

The **numericInput()** function needs to be used to accept only numeric input from a user.

The **textOutput()** and **renderText()** functions need to be used to output text in a Shiny application at the client-side and server-side, respectively.

The **req()** function checks whether the variable has a valid value.

### Multiple choice questions

The function required at the server-side for generating a plot output is:

renderText()

renderPlot()

Both of the above

None of these

The function required at the client-side for generating a text output is:

renderText()

renderPlot()

textOutput()

None of these

The function which checks whether a variable has a legitimate value is:

`req()`

`validate()`

Both of the above

None of these

## Answers to MCQs

B

C

A

## Questions

Modify the SankhyaSaabdh application to also display the number to Words conversion in Hindi language according to the Indian convention.

## Key terms

**HTML:** Hyper Text Mark-up Language

## Shiny Application 2

In [chapter 5: Shiny Application 1, we created a Shiny Application](#) and discussed some features of creating a Shiny application. We discussed how to create layouts of a web page and place components into the layouts. We also discussed how to accept user input and process the input to produce output.

We start this chapter by discussing about RMarkdown. This is another aspect of R programming. Using RMarkdown, we can create dynamic documents as they can contain R code. We will create a simple Shiny application for generating Word Cloud for any text provided through an input file. We had discussed the application for generating a Word Cloud from structured and unstructured data in [Chapter 3: Developing Simple Applications in](#) Data coming from an input file will be unstructured data.

We will discuss several aspects of Shiny programming through this example including how to make use of JavaScript in Shiny applications, creating tabs, inputting files, displaying a progress bar, etc. We will discuss how to handle global variables in a Shiny application. Perhaps most importantly, we will discuss how to make the Shiny application responsive.

Lastly, we will discuss how to integrate a Shiny application with RMarkdown. We will find out how to make Shiny applications

modular. We will also learn the programming style required to make minimum changes when some of the functions of the program need to be changed.

## Structure

In this chapter, we will discuss the following topics:

Creating RMarkdown documents

- Setting up the YAML header
- Creating the body of the RMarkdown
- Embedding R code in RMarkdown
- Generating output from RMarkdown
- Writing our first RMarkdown

Developing the Shiny Application -application ShabdhMegh

- What we will develop
- Creating tabs in a Shiny application
- Creating radio buttons in Shiny applications
- Inputting TEXT files in Shiny applications

- Inputting PDF files in Shiny applications
- Switching modes between TEXT file and PDF file
- What we do before and after a file is input
- Creating output when an input file is provided
- Programming the download button

Complete code of ShabdMegh

## Objectives

After studying this unit, you should be able to:

Create RMarkdown

Create tabs in Shiny applications

Program radio buttons in Shiny applications

Understand how to input files from users in Shiny applications

Use JavaScript in Shiny applications

Use data table in Shiny applications

Understand how to make the Shiny application reactive

Understand how to handle global variables in Shiny applications

Program progress bars in Shiny applications

Integrate Shiny applications with RMarkdown

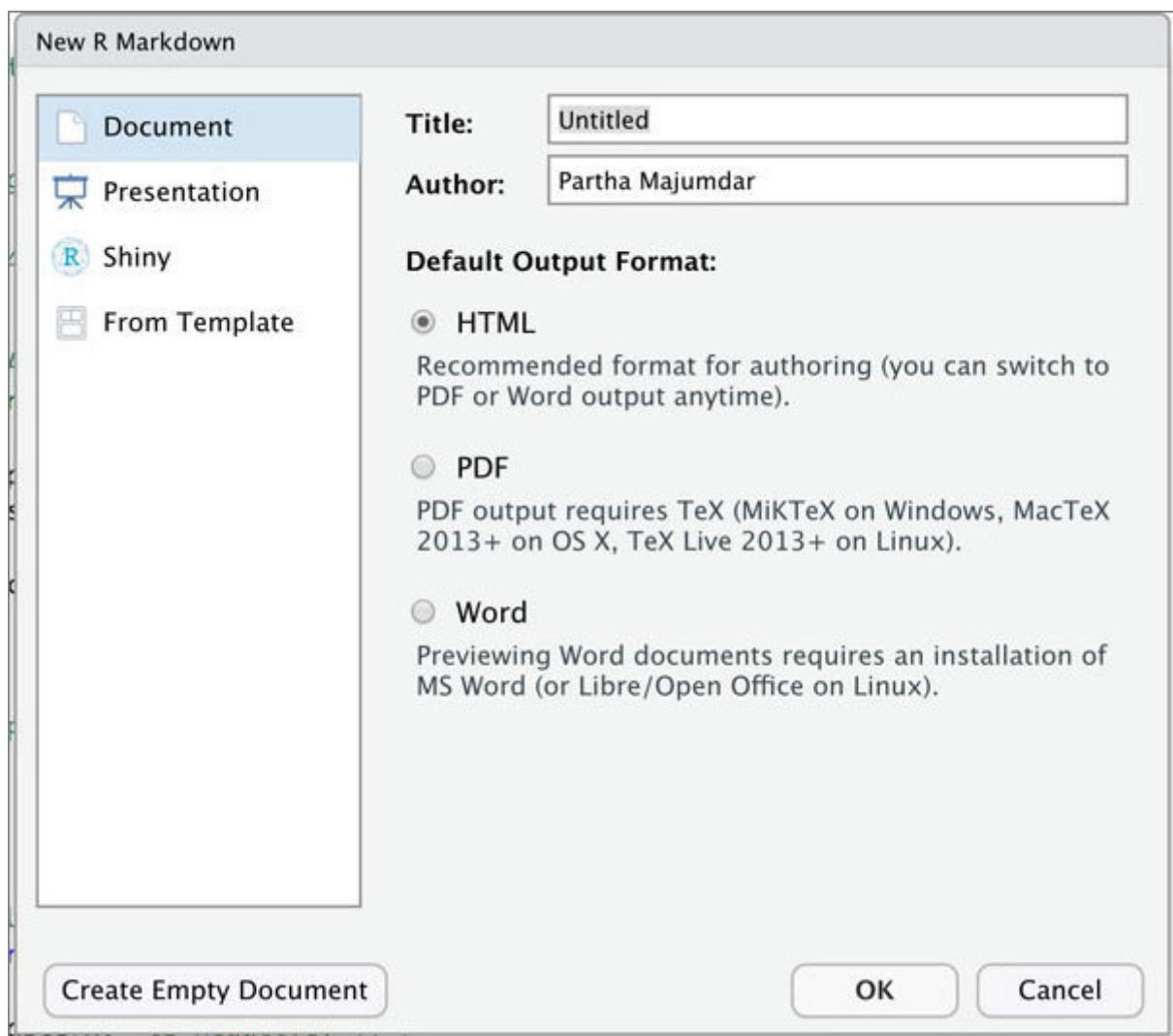
## [Creating RMarkdown document](#)

RMarkdowns are used to create documents in HTML, PDF, and Word document formats. RMarkdowns allow for almost all kinds of formatting options available in any of these document formats. In addition, it is possible to embed R code in these documents. The addition of R code makes these documents dynamic. So, when we create an RMarkdown, it can generate many outputs based on the nature of the underlying data. We will go through the basics of RMarkdown documents.

RMarkdowns can be created using RStudio. In RStudio, use the following menu sequence to create an RMarkdown:

**File | New File | R Markdown...**

On using this menu sequence, RStudio provides the dialogue box as shown in the following screenshot:



**Figure 6.1:** Dialog for creating RMarkdown

Against provide the name of the document. This would appear as the title of the document. The value provided against the Author Name appears in the document as well. The RMarkdown can be used to create documents in three formats – HTML, PDF, and Word document. While creating the RMarkdown, it is possible to specify the format in which the document would be produced by RMarkdown. However, this can be changed after the RMarkdown document has been created. Also, the same RMarkdown can be used to create the output document in either of the three formats.

On providing the title and clicking the **OK** button. The default RMarkdown document is created by RStudio. The RMarkdown has two parts. The first part is the YAML header and the rest of the RMarkdown is used to generate the output document. The YAML header controls the nature of the output to be produced by the RMarkdown.

## Setting up the YAML header

The default YAML header is shown below:

```
---
title: "Untitled"
author: "Partha Majumdar"
date: "07/08/2020"
output: html_document
---
```

Notice that the YAML header is enclosed with two sets of --- (three dashes or hyphens). We can change the values of all the parameters enclosed between ---. We can provide a suitable title as per our document's contents. We can provide the author's name against the tag author.

Against the output tag, we can give three values. You can see that the default value is This produces the output as an HTML document. The HTML output can be used to render the output on any browser. If you provide the value as then the output produced is a PDF document. Similarly, if we provide the value as the output produced is a Word document.

We can set up the RMarkdown to produce the output document in either of these formats. We can change the YAML header as shown below:

```
---
```

```
title: "Emotion Analysis"  
author: "Partha Majumdar"  
date: "07/08/2020"  
output:
```

```
pdf_document:  
fig_caption: yes  
keep_tex: yes  
number_sections: yes  
toc: yes  
html_document: default  
word_document: default
```

```
---
```

Notice that under each document type option, there are more tags available for controlling the output document. We will not discuss these options any further as this is not a book on RMarkdown; rather, we will use RMarkdown for creating the application for emotion analysis.

We will look at one last aspect of the YAML header. In the YAML header, we can provide R code, and this can make the output of the RMarkdown dynamic. One of the very frequent requirements is that we need the RMarkdown to display the date of the output document as the date when the output document is created by the user. We can achieve this by providing the following code against the date tag as shown in the following code:

```
---
```

```
title: "Emotion Analysis"  
author: "Partha Majumdar"  
date: "r format(Sys.Date(), "%B %d, %Y")"  
output:  
pdf_document:  
fig_caption: yes  
keep_tex: yes
```

```
number_sections: yes  
toc: yes  
html_document: default  
word_document: default
```

---

See that the R code starts with '**r (backquote r)**' and the R code ends with a backquote. Within these two markers, we can provide any R code. The code earlier fetches the system date from the server where this code is executed, and formats this as month, date, and year. The function `Sys.Date()` gets the current system date. The `format()` function can be used to format the date.

## [Creating the body of the RMarkdown](#)

Now, let us discuss how we can create the body of the RMarkdown. In the RMarkdown, we can add any free format text as we would like to present in our document. For example, we could add the following text in our RMarkdown:

This is an example of RMarkdown. In this example, we will output the Word Cloud generated from the Resume of Partha Majumdar.

Having written this text, we would like to make the word RMarkdown bold. To do this, we need to enclose the word RMarkdown within a set of \*\* as shown in the following code:

This is an example of **\*\*RMarkdown\*\***. In this example, we will output the Word Cloud generated from the Resume of Partha Majumdar.

All the text enclosed within a set of \*\* would be rendered in bold in the output document. Similarly, to make some text appear in italic, we need to close that text within a set of \* as shown below. We have programmed for the words Word Cloud to appear in italic as shown in the following code:

This is an example of **\*\*RMarkdown\*\***. In this example, we will output the *\*Word Cloud\** generated from the Resume of Partha Majumdar.

We can add headers to the document. If we add the following code in the RMarkdown, this will generate a level 1 header in the output document:

## # Purpose of the Document

This is an example of \*\*RMarkdown\*\*. In this example, we will output the \*Word Cloud\* generated from the Resume of Partha Majumdar.

Notice that the level 1 header is coded by placing a # in the first column of a new line followed by the text to appear as level 1 header. Similarly, if we start a line with ## this would be rendered as a level 2 header. Starting a line with ### would be rendered as a level 3 header and so on:

## # Purpose of the Document

This is an example of \*\*RMarkdown\*\*. In this example, we will output the \*Word Cloud\* generated from the Resume of Partha Majumdar.

## Level 2 Header

### Level 3 Header

#### Level 4 Header

##### Level 5 Header

###### Level 6 Header

These are equivalent to H6 tags in HTML.

## Embedding R code in RMarkdown

Now let us discuss the most interesting aspect of RMarkdown – the embedding of R code in RMarkdown. To embed R code in an RMarkdown, we need to enclose the R code between the following markers as shown in the following code:

```
'''{r}  
'''
```

All R code in RMarkdown starts with '''{r} and ends with Note that these are backquotes. Each piece of R code in an RMarkdown is referred to as a Let us see an example of an R code in an RMarkdown:

```
'''{r most_frequent_words, include=TRUE, echo=TRUE,  
warning=FALSE}  
head(v_df)  
'''
```

Notice that in the start marker of the R code, we have added some extra code. This code can be used to control how the R code would be treated inside the RMarkdown. The first parameter, in this case is the name of the chunk Though it is not mandatory, it is a good practice to name each chunk. This is especially useful when debugging the RMarkdown, or when the RMarkdown becomes a very large piece of code.

The second parameter is **include**. This parameter can take the values of **TRUE** and **FALSE**. When the **include** parameter is set to **TRUE**, the output of the R code in the chunk is included in the output from the RMarkdown. When we set **include** to **FALSE**, the output from the R code in the chunk is not included in the output from the RMarkdown.

The next parameter is **echo**. This parameter can take the values of **TRUE** and **FALSE**. When the **echo** parameter is set to **TRUE**, the R code in the chunk is included in the output generated by the RMarkdown. It is the converse when the **echo** is set to **FALSE**.

The next parameter we will discuss is **warning**. This parameter can take the values of **TRUE** and **FALSE**. When the **warning** parameter is set to **TRUE**, if any warnings are generated by the R code in the chunk, these warnings are included in the output generated by the RMarkdown. The warnings generated by the R code in the chunk are suppressed in the output of the RMarkdown if the **warning** parameter is set to **FALSE**.

Let us see some circumstances of when to set the parameters **include**, **echo**, and **warning** to **TRUE** and **FALSE**. We generally include the following code in every RMarkdown as shown in the following code:

```
'''{r setup, include=FALSE, echo=FALSE, warning=FALSE}
knitr::opts_chunk$set(echo = TRUE)
'''
```

knitr is the R Tool which generates the output from an RMarkdown. We generally refer to generating output from an RMarkdown as Knitting a RMarkdown. knitr parameters can be set programmatically. However, we most possibly would not like any output generated while setting the **knitr** parameters to be included in the output generated by the RMarkdown. So, we would set include as

Similarly, when generating any executive report or in general any report using RMarkdown, we may not want to display the code for setting the knitr parameters in our output document. So, we would in most circumstances set the echo parameter as Generally, no warnings are generated while setting knitr parameters. Even if they were generated, we would most possibly not want to include those warnings in the output that we want to generate from the RMarkdown. So, we would, in most cases, set the warning parameter for this chunk as

When we use R Code, we will generally need to include some libraries. In most cases, it may not be needed to include this code in the output generated by the RMarkdown to show what libraries were used to generate the output. However, in case the output being generated is an academic paper, one may want to include the information of the libraries which were used to generate the output. So, you may decide based on the need.

Following is a sample code for loading libraries in an RMarkdown:

```
'''{r load_libraries, include=FALSE, echo=FALSE, warning=FALSE}
library(wordcloud)
```



## Writing our First RMarkdown

Now that we have gone through the basics of RMarkdown, let us write our first RMarkdown. I will break this code in seven parts. You can put the seven parts together to create complete program. The first part contains the YAML header and the code for the settings:

---

```
title: "R Markdown Demonstration"
author: "Partha Majumdar"
date: "r format(Sys.Date(), "%B %d, %Y)"
output:
pdf_document:
fig_caption: yes
keep_tex: yes
number_sections: yes
toc: yes
html_document: default
word_document: default
```

---

```
'''{r setup, include=FALSE, echo=FALSE, warning=FALSE}
knitr::opts_chunk$set(echo = TRUE)
'''
```

The second part contains the code for loading the required libraries:

```
'''{r load_libraries, include=FALSE, echo=FALSE, warning=FALSE}
library(tm)
```

```
library(tibble)
library(readr)
library(textclean)
library(wordcloud)
'''
```

The third part contains the code for reading the input file and generating the word frequency. All of this R code would be familiar to you as we have discussed this in [Chapter 3: Developing Simple Applications in](#)

```
'''{r read_file, include=FALSE, echo=FALSE, warning=FALSE}
# Read the TXT File
v_text <- tibble(text = gsub("[^[:alnum:]]///'", "", 
read_lines("./Resume.txt")))
# Drop the Empty Lines
v_raw_text <- drop_empty_row(v_text)
'''
```

```
'''{r extract_words, include=FALSE, echo=FALSE, warning=FALSE}
# Create the Corpus
v_corpus <- Corpus(VectorSource(v_raw_text))
```

```
# Create the Term Document Matrix after cleaning the Corpus
```

```

v_tdm <- TermDocumentMatrix(v_corpus,
control =
list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,

bounds = list(global = c(1, Inf))
)
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))
...

```

In the fourth part of the code, we have the code for what should appear in the output document. We have discussed some of these in the previous sections. We will discuss the portions of the code we have not discussed yet:

```
# Document Analysis
```

**\*\*File Name: Resume.txt\*\***

The document provides basic analysis of the file \*Resume.txt\*.

This document is developed to demonstrate generating \*\*R Markdown Documents\*\*.

- \* This program does the following:
  - + Load the needed libraries
  - + Read the File
  - + Compute the Word Frequency after cleaning the data in the file
  - + Form the Word Cloud from the Word Frequency

## ## Most Frequent Words

There are 'r nrow(v\_data)' unique words in the Document.

Given below are the most frequent words used in the Document.

```
""{r most_frequent_words, include=TRUE, echo=TRUE,
warning=FALSE}
head(v_data)""
```

Take a note of the piece of code shown in the following code:

- \* This program does the following:
  - + Load the needed libraries
  - + Read the File
  - + Compute the Word Frequency after cleaning the data in the file
  - + Form the Word Cloud from the Word Frequency

This piece of code is used to generate bulleted lists in RMarkdown. The output of this code is shown in the following:

This program does the following:

- Load the needed libraries
- Read the File
- Compute the Word Frequency after cleaning the data in the file
- Form the Word Cloud from the Word Frequency

Next, we will discuss the feature where we can embed R code in line with the text. I have used one such example in the preceding code as follows:

There are 'r nrow(v\_data)' unique words in the Document.

Notice that the in-line R code is enclosed with 'r and ' (backquote). The output of this code would be something as shown in the following code:

There are 833 unique words in the Document.

Let us examine a piece of code where we include the output from a piece of code in the document. Take a look at the piece of the following code. Notice that we have set include=TRUE and

```
'''{r most_frequent_words, include=TRUE, echo=TRUE,
warning=FALSE}
head(v_data)
'''
```

The output generated is as shown in the following code:

```
head(v_data)
##   values      ind
## 1     71 develop
## 2     61 system
## 3     51 manag
## 4     49 use
## 5     34 solut
## 6     33 project
```

In the fifth part, we have the code for generating the word cloud and the documentation for the same. The documentation explains the expected output:

## ## Word Cloud

Below is the Word Cloud of the most frequent words used in the Document. The Size and the Color of the Words in the Word Cloud is set based on the frequency of the word.

## ### Generating displaying the Code and Suppressing the Warnings

The code used to generate the Word Cloud has been displayed in the document. This can be accomplished by setting the parameter **\*\*echo\*\*** equal to TRUE.

The \*warnings\* can be suppressed by setting the parameter \*\*warning\*\* to FALSE.

```
'''{r generate_word_cloud, include=TRUE, echo=TRUE,
warning=FALSE}
wordcloud(words = v_data$ind, freq = v_data$values, min.freq = 1,
max.words=100, colors=brewer.pal(8, "Dark2"), random.order =
FALSE)
'''
```

In the sixth part of the code, we will once again generate the word cloud. Notice that we have set So, we expect to see the generated warnings:

```
### Generating Word Cloud NOT displaying the Code, but
SHOWING the Warnings
```

```
'''{r generate_word_cloud_2, include=TRUE, echo=FALSE,
warning=TRUE}
```

```
wordcloud(words = v_data$ind, freq = v_data$values, min.freq = 1,
max.words=220, colors=brewer.pal(8, "Dark2"), random.order =
FALSE)
'''
```

Part of the output of this code is as shown as follows. I have only included some of the warnings. Apart from the warnings, the output of this code also includes the word cloud. However, you would be familiar with the word cloud from our discussion in [Chapter 3: Developing Simple Applications in](#)

```
## Warning in wordcloud(words = v_data$ind, freq =
v_data$values, min.freq = 1, :
```

```
## specialist could not be fit on page. It will not be plotted.  
## Warning in wordcloud(words = v_data$ind, freq =  
v_data$values, min.freq = 1, :  
## udemi could not be fit on page. It will not be plotted.  
## Warning in wordcloud(words = v_data$ind, freq =  
v_data$values, min.freq = 1, :  
## Vodafone could not be fit on page. It will not be plotted.
```

The seventh and last piece of code that we will discuss is Latex code. We can include Latex code in RMarkdown. However, all the Latex codes included in the RMarkdown cannot produce the desired output for all types of output generated by RMarkdown:

```
\center  
style="color:red">>**----- THE END -----**  
\center
```

For example, we can include a Latex code for inserting a page break in the output document. However, this code will be effective only if the output format selected for the RMarkdown is either PDF or Word document. In an HTML output, a page break has no meaning.

To include a page break, we can include the following code in the RMarkdown:

```
\newpage
```

Similarly, to include a line break, we can include the following code in the RMarkdown:

```
\newline
```

Anything between the set of **\center** should cause the text in between to be centered on the line. However, this may not be the case always. To understand this code, let us see the output generated by this code when the output format is Word document. It is shown below:

— THE END —

Notice that in the output, the text is not centered on the line. Also, the text is not in red color. These are some issues with a Latex code in RMarkdown. If the same RMarkdown is used to generate the output as a PDF file, you will notice that this output is centered on the line and is in red color. Try it out yourself!

## Generating Output from RMarkdown

So far, we have discussed how to create the RMarkdown. Once the RMarkdown is created, we would like to generate the output from the RMarkdown. RStudio provides a simple way to get this done.

In RStudio, look for the Knit button as shown in the following screenshot:



**Figure 6.2:** Knit button in RStudio

On clicking the **Knit** button, RStudio provides the output **File** options. When one of the output file options is selected, RStudio generates the output from the RMarkdown. While generating the output from RMarkdown, if any error is encountered, RStudio stops the operation.

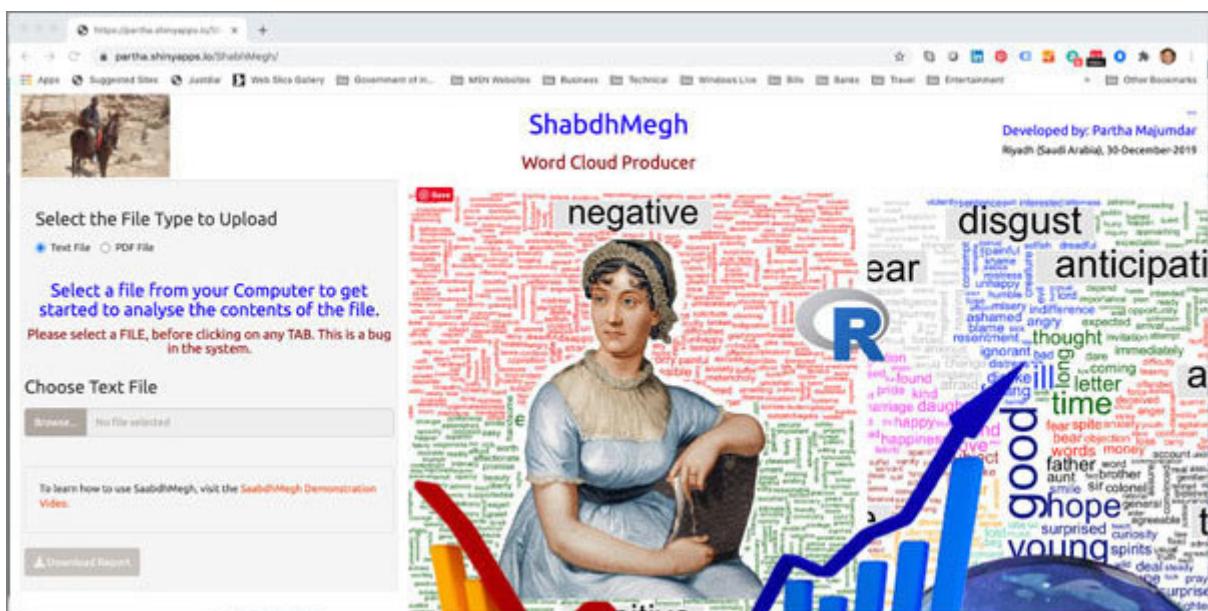
## [\*Developing Shiny application ShabdhMegh\*](#)

Now we will discuss how to create a Shiny Application for generating a word cloud that we discussed in [Chapter 3: Developing Simple Applications in](#) I have named this application ShabdhMegh. In [Chapter 3: Developing Simple Applications in](#) we discussed how to generate the word cloud after reading the contents from a TEXT file. We will add a feature in ShabdhMegh to enable us to read the contents from a PDF file.

## What We will develop?

First, let us discuss what we will develop and then go component by component to develop the same. The application ShabdhMegh is available at

We get the following screen when the application is invoked:



**Figure 6.3:** ShabdhMegh opening screen

In [Chapter 5: Shiny Application](#) we already discussed how to create the layout. I have kept the layout the same for this application as well so that it is easy to relate. On the top, we have the header. In the left-hand side panel, we will take the input, which is the file with contents for which the word cloud has to be generated.

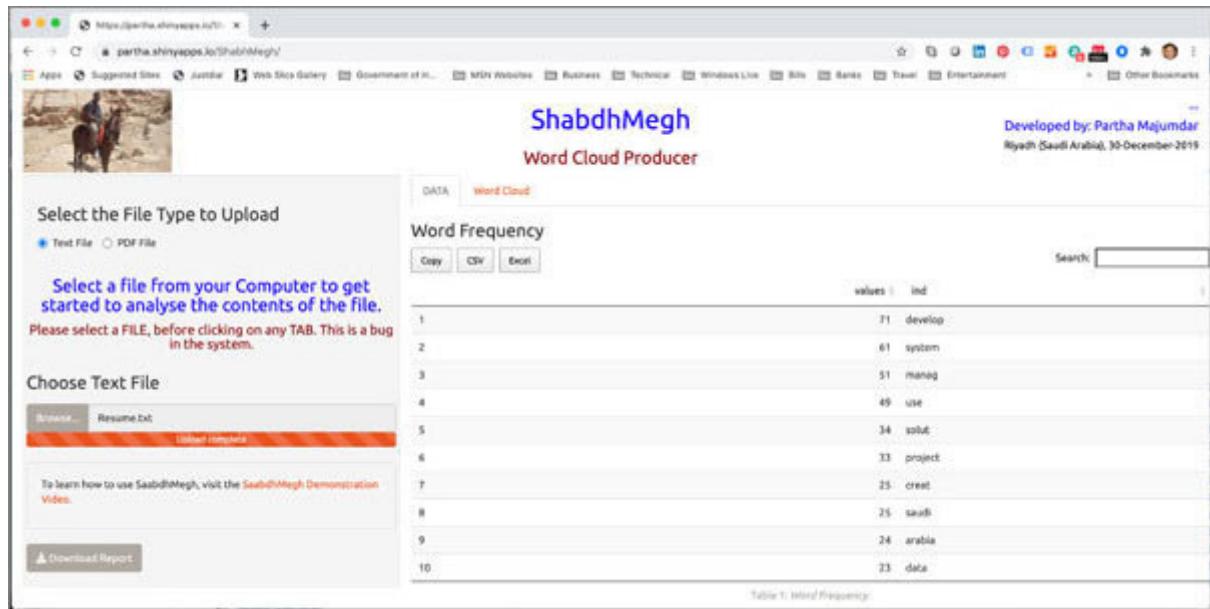
In the right-side panel, we will display the output which is the word cloud.

This time, I have added an extra output in the right-hand panel, which is a static picture displayed when no file has been selected by the user. This is done so that the right-hand panel is not empty when no file has been selected. So, we have a need for the output in the right-hand panel to be changed based on the state of the usage of the application.

We have a similar need in the left-hand panel. In the left-hand panel, we will allow the user to either input a TEXT file or a PDF file. The code required for inputting a TEXT file and for inputting a PDF file are different. Also, the controls we need for this purpose have to be different. So, we have to change the controls for accepting the input file based on whether a TEXT file has to be input, or a PDF file has to be input.

You would have noticed that in the left-hand panel, we have used a radio button to allow the user to choose between a TEXT file and a PDF file. Lastly, notice that in the left-hand panel, we have a button called This button, when clicked, will allow the user to download the report from this application. The report would be created using RMarkdown. So, we will invoke an RMarkdown from the Shiny application. The **Download** button is inactive till an input file is provided. The **Download** button becomes active only when an input file is provided.

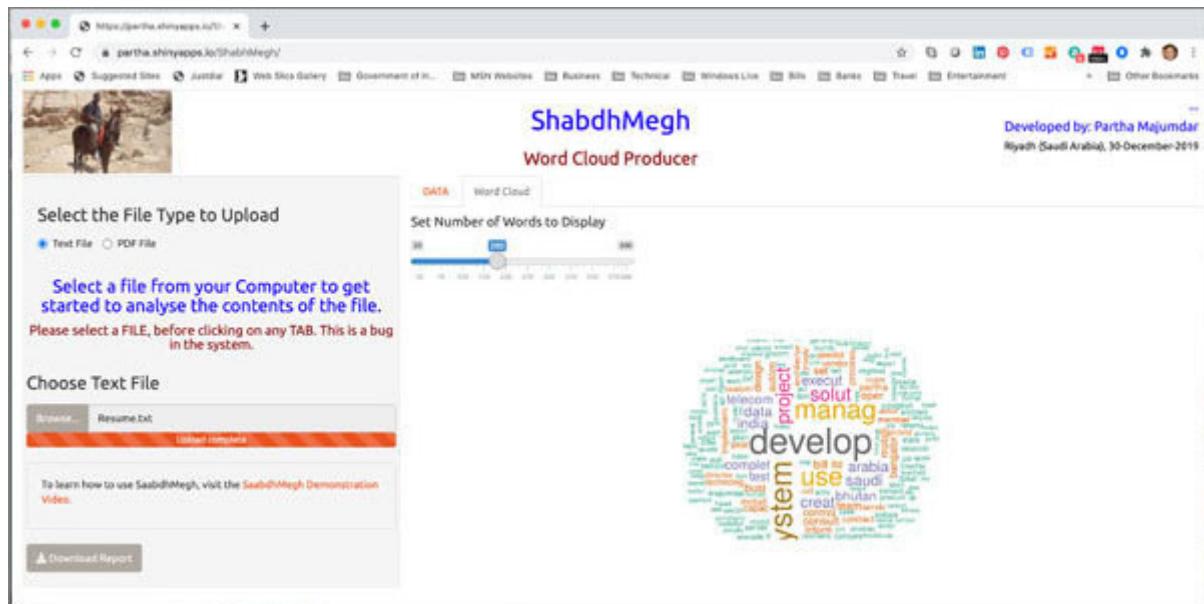
When the user provides an input file, the UI of the application will change as shown in the following screenshot:



**Figure 6.4:** ShabdhMegh UI when a file is provided as input by the user

Notice that in the right-hand panel, we have a tab set with 2 tabs. [Figure 6.4](#) displays the first tab which shows all the words in the input file with their frequencies.

The second tab is shown in the following screenshot which displays the Word Cloud:



**Figure 6.5:** ShabdMegh output with a display of the word cloud

We will go over the code feature by feature and concept by concept.

## [Creating tabs in Shiny applications](#)

The first aspect we discuss is how to create tabs in a Shiny application. To create the tabs, we need to program the front-end of the Shiny application. In other words, the code for creating the tabs needs to be in the **ui()** function.

To create the tabs, we first need to create a **TabSet** panel. The TabSet panel can be created using the **tabsetPanel()** function. We need to provide an identifier for the TabSet panel and for this we use the **id** parameter. So that the TabSet panel can host tabs, we need to define the **type** parameter as equal to **tabs**.

To create the tabs inside in a **TabSet** panel, we need to use the **tabPanel()** function. The first parameter of the **tabPanel()** function is the title of the tab. This value is displayed when the tab is rendered. Following the title, we need to provide the code for all the controls that need to be displayed in the tab. Let us see a code snippet from ShabdMegh where we have created tabs:

```
tabsetPanel(id = "mainTabsetPDF", type = "tabs",
tabPanel("DATA",
h3("Word Frequency"),
dataTableOutput("PDFcontents")
),
tabPanel("Word Cloud",
sliderInput("v_number_of_words_pdf",
```

```

h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudPDF", width = "100%")
)

```

Notice that the identifier for the **TabSet** panel is Also note that this TabSet panel has two Tabs2 tabs with the titles DATA and Word Cloud. Inside the DATA tab, we display a level 3 header and the output of a data table. Inside the Word Cloud tab, we display a slider and the output of a plot.

Shown in the following screenshot is one of the outputs produced by this code. Notice the tabs:

	values	Ind
1	71	develop
2	61	system
3	51	manag
4	49	use
5	34	solut
6	33	project
7	25	creat
8	25	saudi
9	24	arabia
10	23	data

Table 1: Word Frequency.

**Figure 6.6:** Tabs in ShabdhMegh



## [Creating radio button in Shiny applications](#)

To create radio buttons in Shiny application, we can use the function Please see the related code from ShabdhMegh before we discuss this further:

```
radioButtons("v_file_type", h3("Select the File Type to Upload"),  
c("Text File" = C_TEXT_FILE_INDICATOR,  
"PDF File" = C_PDF_FILE_INDICATOR),  
inline = TRUE),
```

Radio buttons are used for taking input from the users. So, this input must be passed on to the server-side code of the Shiny application for the appropriate actions. So, we need an identifier to be able to identify the input. The first parameter of the **radioButtons()** function is the input identifier. In our case, the identifier is named as So, we can capture the input provided by the user through the radio button in the server-side code by referencing the variable

The second parameter is the label which is displayed in the UI. The label helps the user understand what input is being captured through the radio buttons. The third parameter of the **radioButtons()** function is the set of options to display under this radio buttons. In the preceding example, we provided two options – Text file and PDF file. Notice that the options are provided as a

vector. Further notice that we have written "Text File" = Here, Text file is what is displayed in the UI. However, when the text file option is selected by the user through the radio button, the value of radio button is set to the value of The variable input\$v\_file\_type will either have the value C\_TEXT\_FILE\_INDICATOR or

C\_TEXT\_FILE\_INDICATOR and C\_PDF\_FILE\_INDICATOR are constants declared in the program as shown in the following code:

```
C_TEXT_FILE_INDICATOR <- "TXT"  
C_PDF_FILE_INDICATOR <- "PDF"
```

**These constants are used to avoid hardcodeding in the program.** We will soon see that we need to evaluate which option was selected by the user through the radio button in the server-side code. To evaluate the same, we would need writing the values TXT and/or PDF once again. If we did so, and if we changed any of the values and/or we added one or more option(s), we would have to visit all the parts of the code where we would have used these hardcoded values.

**As a general practice, any hardcodeding should be avoided in any program.**

The last parameter to the radioButtons() function is inline. When this parameter is set to the radio button options are displayed in the same line. When the inline parameter is set to the radio button options are displayed one below the other. Shown in the following screenshot is the output produced by this code:

# Select the File Type to Upload

- Text File  PDF File

*Figure 6.7: Radio Buttons in ShabdhMegh*

## [\*Inputting TEXT files in Shiny applications\*](#)

Next, we will discuss how to input a text file from the user in a Shiny application. The code for the same is provided below:

```
fileInput("v_input_file_txt", h3("Choose Text File"),  
multiple = FALSE, placeholder = "No file selected",  
accept = c("text/csv",  
"text/comma-separated-values,text/plain",  
.txt")  
,
```

The function `fileInput()` is used for inputting a file from the user. In Shiny Applications, the function **fileInput()** is used for inputting any type of file. It is possible to restrict the type of file that will be allowed for inputting through the parameters of the function

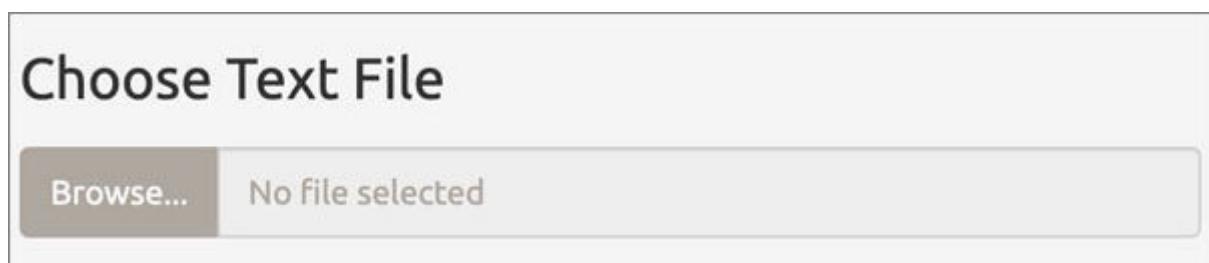
Let us go through the parameters of the function. The first parameter is the variable that we will associate with this control. We already know that as we will be using this control for taking an input, we should be able to refer to the input value through a variable. In the earlier example, the variable associated with this control is So, we can refer to this variable in the server-side code as We could have written this code as `inputId = v_input_file_txt` to make it explicit.

The second parameter is the label that we will display to the user for providing guidance on what this control is being used for in the application. Notice that in the example code provided earlier, we have used a level 3 HTML Header. We could have made this code explicit by writing **label =**

The third parameter multiple controls whether we will allow input of one file or more than one file. Setting `multiple = FALSE` allows input of only one file at a time. The next parameter placeholder displays a static text in the control when no file has been input by the user.

The last parameter we will discuss is accepted. Using this parameter, we control what type(s) of file(s) we will allow for inputting by the user. In the preceding example, the setting of the accept parameter only allows the input of TXT and CSV files.

The preceding code produces the following UI:



**Figure 6.8:** Text file input in ShabdMegh

When the user clicks the **Browse** button, the file explorer is opened and the user can select a file. Due to the settings of the accept parameter, the file explorer will only allow the selection of

TXT and CSV files in this case. Files of all other types will be greyed out.

## Server-side Code side code for inputting text file

Let us see the code we need at the server-side for inputting a text file:

```
v_text <- tibble(text = gsub("[^[:alnum:]]///'", "",  
read_lines(input$v_input_file_txt$datapath)))
```

We are familiar with this code as we have discussed this in [Chapter 3: Developing Simple Applications in](#). However, notice the parameter for the function. We know that we need to pass the file name to the function. Also, we know that in the UI, we have accepted the TXT file in the variable. So, the variable `v_input_file_txt` can be referred to in the server-side as. However, we need to pass the complete file path of the input file to the function. So, we need to pass which provides the complete path of the input file.

When we load a file from the local machine, we know the exact path of the file. However, when we load a file in a Shiny application, the file is loaded from the local machine to the server from where the Shiny application is executing. So, we cannot know the exact path of the file once it is loaded from the local machine to the server. The only way to get the complete file path is to use the variable `datapath` of the variable referencing the file.

## Inputting PDF files in Shiny applications

Inputting PDF files is very similar to inputting TEXT files as we need the function The only change is to change the value for the accept parameter as shown in the following code:

```
fileInput("v_input_file_pdf", h3("Choose PDF File"),  
multiple = FALSE, placeholder = "No file selected",  
accept = c("application/PDF",  
".pdf")  
)
```

The output produced is as shown in the following screenshot:



**Figure 6.9:** PDF file input in ShabdhMegh

## Server-side Code for inputting PDF Files

For uploading PDF files, we need the library So, we need to upload the library as follows:

```
library(pdftools)
```

The server-side code for uploading a PDF file is as follows. Notice that we need to pass the full path of the input file to the function pdf\_text() for uploading the PDF file:

```
v_raw_text <- pdf_text(input$v_input_file_pdf$datapath) %>%
strsplit("\n")
```

Once the PDF file has been uploaded, we need to split each line of the text in the PDF file into separate vectors. We can accomplish this by using the function strsplit(). \n is used to indicate the new line character. You should be familiar with the pipe operator %>% and we know that to use this operator, we need to load the library

## Switching Modes between TEXT file and PDF file

We have seen how to accept a TXT file and a PDF file as input. However, we must only accept one file as input. The input file has to either be a TXT file or a PDF file. So, we must switch between the type of file that should be input based on the selection of the file type as indicated by the user through the radio button.

To program this feature, we need JavaScript. To use JavaScript in Shiny application, we need the library named So, the first thing to do in the Shiny application is to upload this library application as follows:

```
library(shinyjs)
```

Next, we must include this piece of code in the ui() function of the Shiny application:

```
ui <- fluidPage(  
  shinyjs::useShinyjs(),
```

**shinyjs::useShinyjs()** should be the first line of code in the ui() function. If this is not the case, none of the JavaScript code, that we will discuss, will work in the Shiny application. The Shiny application will work otherwise; however, there will be no action from any of the JavaScript code.

We will now examine the code for switching between a TXT file and a PDF file.

**All the following code being discussed should be written in the server() function. This is because any computation must take place in the server and the output/decision must be reflected in the client (or UI).**

```
if (input$v_file_type == C_TEXT_FILE_INDICATOR) {  
  shinyjs::hide("v_input_file_pdf")  
  shinyjs::show("v_input_file_txt")  
} else {  
  shinyjs::show("v_input_file_pdf")  
  shinyjs::hide("v_input_file_txt")  
}
```

We know that the radio button control was assigned the variable So, we check its value of input\$v\_file\_type to decide whether to enable a TXT file input or a PDF file input. We have also discussed the use of the constants C\_TEXT\_FILE\_INDICATOR and You can see them being used here.

Notice the use of the functions shinyjs::hide() and These two functions are used to hide and show a control in Shiny Applications, respectively. The control to hide or show is passed as a parameter to these functions. So, the code shinyjs::hide(v\_input\_file\_pdf) will hide the PDF file input control.

So, based on the file type chosen by the user, the TXT file input control and the PDF file input control are either hidden or shown.

However, this code must get executed every time we make a change to the choice in the radio button control. To achieve this, we must use the `observe()` function as shown in the following code:

```
observe({
  if (input$v_file_type == C_TEXT_FILE_INDICATOR) {
    shinyjs::hide("v_input_file_pdf")

    shinyjs::show("v_input_file_txt")
  } else {
    shinyjs::show("v_input_file_pdf")
    shinyjs::hide("v_input_file_txt")
  }
})
```

The `observe()` function takes a code block as one of the parameters. `observe()` can take another parameter as a control to observe. We will see this later. The `observe()` function, without any control being mentioned, executes the code block when there is any change in any of the controls in the UI. So, whenever the user changes the choice in the radio button, this code block executes and hides or shows the associated control for inputting the file.

Notice that the preceding code is not complete. We will discuss this in the next section as we have more actions to perform when the radio button choice is changed.

## What we do before and after a file is input?

We discussed earlier that when the application is opened, it displays a static picture. Let us see the code for getting this done:

```
img(src="CourseLogo.jpg", width=1200, height=900)
```

We know from [Chapter 5: Shiny Application 1](#) how to display an image. The preceding code should be enough. Here, our image file is However, we would like to hide and show the image based on the state of the application. As the image is not a control, we cannot do the activity with just this code. For achieving this function, we need a variable to refer to for hiding or showing. We can achieve this by enclosing the preceding piece of code in a

We can write this code as shown in the following code:

```
div(id = "DummyDisplay",
img(src="CourseLogo.jpg", width=1200, height=900),
),
```

Here, the **div** provides us a reference able variable with the id as

In the same fashion, we need to enclose the code for creating the tabs within divs tag as shown below. We will create one div for

displaying the output when the input file is a PDF file We will name this as

```
div(id = "OutputTabsPDF",
tabsetPanel(id = "mainTabsetPDF", type = "tabs",
tabPanel("DATA",
h3("Word Frequency"),

dataTableOutput("PDFcontents")
),
tabPanel("Word Cloud",
sliderInput("v_number_of_words_pdf",
h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudPDF", width = "100%")
)
)
),
```

We will create another div for displaying the output when the input file is a TXT file. We will name this as

```
div(id = "OutputTabsTXT",
tabsetPanel(id = "mainTabsetTXT", type = "tabs",
tabPanel("DATA",
h3("Word Frequency"),
dataTableOutput("TXTcontents")
),
tabPanel("Word Cloud",
```

```

sliderInput("v_number_of_words_txt",
h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudTXT", width = "100%")
)

)
)

```

Now, let us take a look at our complete code for the observe() function being discussed in the previous section. It is shown in the following code:

```

observe({
if (input$v_file_type == C_TEXT_FILE_INDICATOR) {
shinyjs::hide("v_input_file_pdf")
shinyjs::show("v_input_file_txt")
} else {
shinyjs::show("v_input_file_pdf")
shinyjs::hide("v_input_file_txt")
}
if ((is.null(input$v_input_file_pdf) || input$v_input_file_pdf == ""))
&&
(is.null(input$v_input_file_txt) || input$v_input_file_txt == "")
)
{
shinyjs::show("DummyDisplay")
shinyjs::hide("OutputTabsPDF")
shinyjs::hide("OutputTabsTXT")
}

```

```
shinyjs::disable("report")
} else {
if(input$v_file_type == C_TEXT_FILE_INDICATOR) {
if (is.null(input$v_input_file_txt) || input$v_input_file_txt == "") {
}

shinyjs::hide("OutputTabsTXT")
shinyjs::show("DummyDisplay")
shinyjs::disable("report")
} else {
if (v_prev_file_type == C_PDF_FILE_INDICATOR) {
shinyjs::hide("OutputTabsTXT")
shinyjs::show("DummyDisplay")
shinyjs::disable("report")
} else {
shinyjs::show("OutputTabsTXT")
shinyjs::hide("DummyDisplay")
shinyjs::enable("report")
}
}

shinyjs::hide("OutputTabsPDF")
v_prev_file_type <- C_TEXT_FILE_INDICATOR
} else {
if (is.null(input$v_input_file_pdf) || input$v_input_file_pdf == "") {
shinyjs::hide("OutputTabsPDF")
shinyjs::show("DummyDisplay")
shinyjs::disable("report")
} else {
if (v_prev_file_type == C_TEXT_FILE_INDICATOR) {
shinyjs::hide("OutputTabsPDF")
shinyjs::show("DummyDisplay")
```

```

shinyjs::disable("report")
} else {

shinyjs::show("OutputTabsPDF")
shinyjs::hide("DummyDisplay")
shinyjs::enable("report")
}
}

shinyjs::hide("OutputTabsTXT")
v_prev_file_type <- C_PDF_FILE_INDICATOR
}
}
})

```

Notice that when the input file for TXT file and PDF file is not provided, we display the image. We check for whether the input file is provided or not by checking whether the variable for input file is null or not. To check for null, we use the function

Notice the codes shinyjs::enable("report") and report is the name of the variable we have used to refer to the **Download** button. It is declared as follows. We will discuss this in detail a little later:

```
downloadButton("report", "Download Report")
```

The shinyjs::enable() function is used to enable control. When a control is enabled, the user can see and control uses the control. Otherwise, the user can only see it. And the shinyjs::disable() function is used to disable a control.

Lastly, we will study the code pertaining to the variable We declare this variable as a global variable right at the beginning of the Shiny application as follows:

```
v_prev_file_type <- C_TEXT_FILE_INDICATOR
```

See that we have set to a default value to indicate that TXT file is the chosen file type. This is because the default value of the radio button is to select a TXT file type input. We use this variable to check for a valid input file having been provided when we change the value of the radio button. So, every time we change the radio button selection, we need to update this variable to indicate what was the last file type chosen by the user. So, this variable must be updated in the observe() function that we are discussing in this section. However, this observe() function is a function in the server() function. In other words, this is a local function in the function

So, here if we write the following code to update the variable it will treat v\_prev\_file\_type as a local variable and thus the global variable will not get updated:

```
v_prev_file_type <- C_TEXT_FILE_INDICATOR
```

To update the global variable, we need to use the operator <<- as shown in the following code:

```
v_prev_file_type <<- C_TEXT_FILE_INDICATOR
```

## [Creating output when an input file is provided](#)

When we input a file, we will create two outputs. The first output is a list of words in the file along with their frequencies. A word cloud is the second output that we will create. As we have discussed before, we will put these two outputs<sup>2</sup> outputs in two separate tabs. We have named the tab for displaying the word frequency as And we have named the tab for displaying the word cloud as Word Cloud.

The **tabsetPanel** to display when the input file type is PDF file is as shown in the following code. Notice that the **tabsetPanel** is identified as

```
div(id = "OutputTabsPDF",
tabsetPanel(id = "mainTabsetPDF", type = "tabs",
tabPanel("DATA",
h3("Word Frequency"),
dataTableOutput("PDFcontents")
),
tabPanel("Word Cloud",
sliderInput("v_number_of_words_pdf",
h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudPDF", width = "100%")
)
```

)

),

Also, notice that we have a dataTableOutput and it is identified as We will discuss dataTableOutput in the next subsection. And for identifying the plotOutput for displaying the word cloud, we have used the variable Lastly, we have identified the sliderInput as

Now, let us see the code for creating the tabsetPanel for displaying the output for input file type is TXT file:

```
div(id = "OutputTabsTXT",
tabsetPanel(id = "mainTabsetTXT", type = "tabs",
tabPanel("DATA",
h3("Word Frequency"),
dataTableOutput("TXTcontents")
),
tabPanel("Word Cloud",
sliderInput("v_number_of_words_txt",
h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudTXT", width = "100%")
)
)
)
```

Take a note of the variable names. They are mainTabsetTXT, TXTcontents, and Notice that all the variable names are different

here. This is because, in a Shiny application, we cannot have two different controls having the same name.

**Every control in a Shiny application must have a unique name.**

## Computing the word frequency

Before we can display the word frequency and the word cloud, we need to compute the word frequency from the provided input file. For doing this, we can use the function as shown below. This function is written in the server-side and is thus available within the function

```
calculateWordFrequency <- function(l_text) {  
  v_data <- reactive({  
    # Create the Corpus  
    v_corpus <- Corpus(VectorSource(l_text))  
  
    # Create the Term Document Matrix after cleaning the Corpus  
    v_tdm <- TermDocumentMatrix(v_corpus,  
      control =  
        list(removePunctuation = TRUE,  
             stopwords = TRUE,  
             tolower = TRUE,  
             stemming = TRUE,  
             removeNumbers = TRUE,  
             bounds = list(global = c(1, Inf))))  
  })  
  
  # Find the Frequent Words  
  v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)
```

```

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame
stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))}

return (v_data())
}

```

This function should be familiar as we have discussed it in [Chapter 3: Developing Simple Applications in](#). So, I will not discuss it again. However, there is one aspect which is new and that is the use of the function `Notice`. Notice that the complete code for extracting the word frequency is enclosed within the `reactive()` function. We use the `reactive()` function in a Shiny application when we want a code block to be executed whenever there is any change in the input to any control in the Shiny application.

Notice that we assign the output from the `reactive()` function to the variable `v_so`. So, the contents of this variable are a function. So, to refer to the contents of the variable we need to write

Now, we need to use the function `calculateWordFrequency()` in two contexts – once when the input file is a TXT file and once when the input file is a PDF file. Let us see how we can use this function in the two contexts. We use the following code when the input file is a TXT file:

```
v_df <- reactive({
```

```

# Read the TXT File
v_text <- tibble(text = gsub("[^[:alnum:]]///'", "", 
read_lines(input$v_input_file_txt$datapath)))

# Drop the Empty Lines

v_raw_text <- drop_empty_row(v_text)

# Calculate Word Frequency
calculateWordFrequency(v_raw_text)
})

```

This code should be understandable now. Notice that **v\_df** is a global variable. To get the word frequency, we need to refer to the variable We use the following code when the input file is a PDF file:

```

v_df <- reactive({
# Read the PDF File
v_raw_text <- pdf_text(input$v_input_file_pdf$datapath) %>%
strsplit("\n")
# Calculate Word Frequency
calculateWordFrequency(v_raw_text)
})

```

You would have just noticed an example of how to make the code modular. We made the function `calculateWordFrequency()` generic so that we could use the same function for any type of file taken as input. We manipulated the input to this function so that the function could work seamlessly in all situations.

## **Creating the data table output**

Now that we have computed the word frequency, we have to display the words and their frequencies. A large file will have a lot of words. To display these words in a user-friendly user interface, we need features so that we can paginate the output and allow for scrolling through the output page by page. Further, we will need features like searching for specific words, filtering our words of certain patterns, etc.

A data table provides all these features and much more. We can provide all these features through a very few lines of code using the data table. To be able to code for a data table, we need to load the library **DT** as shown in the following code:

```
library(DT)
```

The following code generates the data table:

```
DT::datatable(  
{v_df()},  
caption = htmltools::tags$caption(  
style = 'caption-side: bottom; text-align: center;',  
'Table 1: ', htmltools::em('Word Frequency.')),  
  
extensions = 'Buttons',
```

```
options = list(  
    fixedColumns = TRUE,  
    autoWidth = TRUE,  
    ordering = TRUE,  
    dom = 'Bftsp',  
    buttons = c('copy', 'csv', 'excel')  
)
```

The function `datatable()` in the DT library generates the data table. The `datatable()` function has a lot of options. We will discuss the minimal set that we need.

### **For a detailed study of the DT library, you could refer to the URL**

The first parameter we need to pass to the `datatable()` function is a data frame that contains the data to display in the data table. We know that the word frequency can be obtained from the data frame

For understanding the remaining parameters of the `datatable()` function, let us see the output generated by this code:

Word Frequency	
<a href="#">Copy</a>	<a href="#">CSV</a>
<a href="#">Excel</a>	1
	2
	3
	4
	5
	6
	7
	8
	9
	10
71	develop
61	system
51	manag
49	use
34	solut
33	project
25	creat
25	saudi
24	arabia
23	data

Table 1: Word Frequency.

Previous 1 2 3 4 5 ... 84 Next

**Figure 6.10:** Data Table output

Notice that the columns are displayed in the data table as marked by the indicator number 3. By default, the column names are the same as that in the data frame. Also, the order of the columns is the same as in the data frame. However, we will see in [Chapter 11: Emotion Analysis on Twitter Data how](#) to provide column names and order the columns as per our preference.

Notice the buttons marked by the indicator number 1. This is produced by the parameter **extensions = "Buttons"** Further, we have specified in the options parameter that we need the buttons for copy, CSV, and Excel through the code **buttons = c("copy", "csv", "excel")**. These buttons can be used to copy the current state of the data table, save the data table to a CSV file and save the data table to an Excel file respectively. There are other buttons as well, like a button to produce the PDF output of the data table or a button to print the data table, etc.

The data table, by default, provides the buttons to scroll through the pages as shown by the indicator number 5. By default, the data table displays 10 records per page. However, we can set the number of records to display per page, which is something we will see in [Chapter 11: Emotion Analysis on Twitter](#)

*The data table, by default, provides the search bar as shown by the indicator number 2.* This can be used for searching for a specific value or for filtering the displayed data based on a provided pattern. The following code produces the caption for the data table. The caption is marked in [figure 6.10](#) by the indicator number 4. Notice that the caption is at the bottom and it is centered. Also, notice that the words word frequency is in italic:

```
caption = htmltools::tags$caption(  
  style = 'caption-side: bottom; text-align: center;',  
  'Table 1: ', htmltools::em('Word Frequency.'))  
,
```

Now that we have discussed the datatable() function, let us see the code we need for rendering the data table in the Shiny application. First, we will write a function as follows:

```
displayDataTable <- function() {  
  DT::datatable(  
    {v_df()},  
    caption = htmltools::tags$caption(  
      style = 'caption-side: bottom; text-align: center;',
```

```

'Table 1: ', htmltools::em('Word Frequency')),

extensions = 'Buttons',

options = list(
fixedColumns = TRUE,
autoWidth = TRUE,
ordering = TRUE,
dom = 'Bftsp',
buttons = c('copy', 'csv', 'excel')))
}

```

So, now we have a single function named `displayDataTable()` to produce the data table. Again, we have made our code modular. However, to render the data table, we need to link the server-side code to the client-side variable.

We have already seen that to display the data table in the client-side when we have a PDF input file, we have the following code as shown in the following code:

```
dataTableOutput("PDFcontents")
```

The function `dataTableOutput()` produces the data table output in the client-side. So, to render the data table in the UI, we need to return the data table in the variable. The code is as shown in the following code:

```
output$PDFcontents <- renderDataTable({
```

```
req(input$v_input_file_pdf)

v_df <- reactive({
  # Read the PDF File
  v_raw_text <- pdf_text(input$v_input_file_pdf$datapath) %>%
    strsplit("\n")
  # Calculate Word Frequency
  calculateWordFrequency(v_raw_text)})
  displayDataTable()
})
```

Notice that, in the server-side, we need the function `renderDataTable()` to render the data table. We have seen the use of the `req()` function in [Chapter 5: Shiny Application](#). The presence of the `req()` function prevents the rest of the code in the function from being executed when there is no input file.

Similarly, for displaying the data table when the input file is a TXT file, we have the following code on the client-side:

```
dataTableOutput("TXTcontents")
```

And we have the following code on the server-side:

```
output$txtcontents <- renderDataTable({
  req(input$v_input_file_txt)
```

```
v_df <- reactive({  
  # Read the TXT File  
  v_text <- tibble(text = gsub("[^[:alnum:]]//'", "",  
    read_lines(input$v_input_file_txt$datapath)))  
  
  # Drop the Empty Lines  
  v_raw_text <- drop_empty_row(v_text)  
  
  # Calculate Word Frequency  
  calculateWordFrequency(v_raw_text)  
})  
  
displayDataTable()  
})
```

## Displaying the Word Cloud

We have seen in [Chapter 3: Developing Simple Applications in R](#) how to generate a word cloud. We will use the same code for generating the word cloud as shown in the following code:

```
wordcloud(words = v_df()$ind, freq = v_df()$values, min.freq = 1,  
max.words = l_number_of_words,  
random.order=FALSE,  
colors=brewer.pal(8, "Dark2")  
)
```

We see that the data comes from The only aspect to notice is that the parameter max.words is set by a variable. This variable gets its value from the slider we have placed for the user to define how many words to include in the word cloud.

So, we need a function to generate the word cloud. We can define the function generateWordCloud() as shown in the following code:

```
generateWordCloud <- function(l_number_of_words) {  
  withProgress(message = 'Generating',  
  detail = 'This may take a while...', value = 0, {  
    wordcloud(words = v_df()$ind, freq = v_df()$values, min.freq = 1,  
    max.words = l_number_of_words,  
    random.order=FALSE,
```

```
colors=brewer.pal(8, "Dark2")
)
})
}
```

We pass the variable `l_number_of_words` as a parameter. We can call the following function when the input file is a TXT file:

```
output$wordCloudTXT <- renderPlot({
req(input$v_input_file_txt)

generateWordCloud(input$v_number_of_words_txt)
})
```

So, we must have an output variable `wordCloudTXT` in the client-side code. And we must have an input variable `v_number_of_words_txt` in the client-side code. If we examine the corresponding client-side code, what we find is as shown in the following code:

```
tabPanel("Word Cloud",
sliderInput("v_number_of_words_txt",
h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudTXT", width = "100%")
)
```

Similarly, when the input file is a PDF file, the server-side code is as shown in the following code:

```
output$wordCloudPDF <- renderPlot({  
  req(input$v_input_file_pdf)  
  
  generateWordCloud(input$v_number_of_words_pdf)  
})
```

And the corresponding client-side code is as follows:

```
tabPanel("Word Cloud",  
  sliderInput("v_number_of_words_pdf",  
    h4("Set Number of Words to Display"),  
    min = 20, max = 500,  
    value = C_SLIDER_DEFAULT_VALUE,  
    step = 10, ticks = TRUE),  
  plotOutput("wordCloudPDF", width = "100%")  
)
```

## Displaying the progress bar

When the Shiny application generates the word cloud, the program could take some time. So that the user does not get an impression that the program is hanging, we can display a progress bar to keep the user engaged. To do so, we can use the function `withProgress()` in the server-side code as shown in the following code:

```
generateWordCloud <- function(l_number_of_words) {  
  withProgress(message = 'Generating',  
              detail = 'This may take a while...', value = 0, {  
    wordcloud(words = v_df()$ind, freq = v_df()$values, min.freq = 1,  
              max.words = l_number_of_words,  
              random.order=FALSE,  
              colors=brewer.pal(8, "Dark2")  
  })  
}  
}
```

With this code, when the word cloud is getting generated, a dialog box will be displayed in the UI with the message Generating and This is may take a However, in the client-side code, we need to specify how this dialog box should appear. So, in the client-side, we need to include this piece of code as shown in the following code:

```
tags$head(  
tags$style(  
HTML(".shiny-notification {  
position:fixed;  
  
top: calc(50%);  
left: calc(50%);  
font-family:Verdana;  
fontSize:xx-large;  
}  
"  
)  
)  
,
```

This code ensures that the dialog box appears centered on the screen.

### Resetting output when a new input file is provided

The last piece of detail regarding generating the output is that we need to cater to the situation when the input file is changed. In this particular case, we would like that the output display should revert to show the **DATA** tab and the slider on the **Word Cloud** tab should get reset. We can achieve the same with the following code:

```
observeEvent(input$v_input_file_pdf, {  
  # Reset Display to DATA TAB  
  updateTabsetPanel(session, "mainTabsetPDF", selected = "DATA"  
}  
  
# Reset the Slider  
updateSliderInput(session, 'v_number_of_words_pdf', value =  
C_SLIDER_DEFAULT_VALUE)  
}  
  
observeEvent(input$v_input_file_txt, {  
  # Reset Display to DATA TAB  
  updateTabsetPanel(session, "mainTabsetTXT", selected = "DATA"  
}  
  
# Reset the Slider  
updateSliderInput(session, 'v_number_of_words_txt', value =  
C_SLIDER_DEFAULT_VALUE)
```

)

First notice that we use the function observeEvent() to observe a particular variable. We have programmed two observeEvent() functions to observe input\$v\_input\_file\_pdf and So, whenever the values in these variables are changed, the corresponding function is fired.

Inside the observeEvent() function, we use the updateTabsetPanel() function and the updateSliderInput() functions. The updateTabsetPanel() function is used to set the selected tab to the DATA tab in the Tabset panels mainTabsetTXT and The updateSliderInput() function is used to set the sliders – v\_number\_of\_words\_txt and v\_number\_of\_words\_pdf - to a default value as defined by the constant

### *Programming the download button*

Lastly, we would program the feature that will enable the user to save the word cloud generated for the input file in an output file. We know that we can use RMarkdown to generate an output file with the word cloud. So, we need to interface RMarkdown with the Shiny application. To achieve this, we need to program the client-side and of the Shiny application, the server-side of the Shiny Application and also the application and the RMarkdown. So, let us proceed one by one.

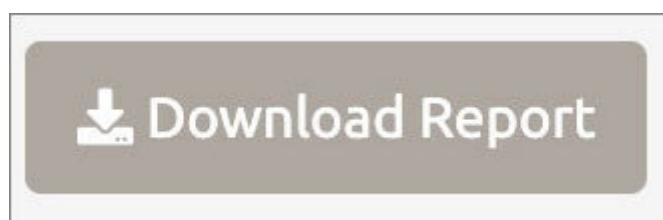
## Programming the client-side of the Shiny application

In the client-side code, we need to declare the **Download** button. We can do so with the following code:

```
downloadButton("report", "Download Report")
```

The downloadButton() function creates the **Download** button. The first parameter of this function is the variable to associate with this button. In this case, the variable is named report. The second parameter is the text required to be displayed on this button. In this case, it is download report.

So, this code produces the following in the UI:



**Figure 6.11:** Download button

## Programming the server-side of the Shiny application

In the server-side, we need to program for what needs to happen when the download button is clicked. What we need is that when the download button is clicked, the Shiny application must call the RMarkdown so that the report is generated and saved to a file. We can achieve this with the following code shown:

```
output$report <- downloadHandler(  
  # For PDF output, change this to "report.pdf"  
  filename <- "report.pdf",  
  
  content <- function(file) {  
    withProgress(message = 'Generating Report. This takes a minute  
    or two...', value = 0, {  
      if (input$v_file_type == C_TEXT_FILE_INDICATOR) {  
        l_file_name <- input$v_input_file_txt$name  
        l_words_to_display <- input$v_number_of_words_txt  
      } else {  
        l_file_name <- input$v_input_file_pdf$name  
        l_words_to_display <- input$v_number_of_words_pdf  
      }  
  
      # Set up parameters to pass to Rmd document  
      v_params <- list(  
        p_input_file = l_file_name,  
        p_number_of_words_to_display = l_words_to_display,
```

```
p_word_frequency = v_df()
)

rmarkdown::render("ShabdhMegh.Rmd",
output_file = file,
params = v_params,
envir = new.env(parent = globalenv())
)
})
}
)
```

In the server-side, we need to program the `downloadHandler()` function. This function takes two parameters – `filename` and `content`. For the `filename` parameter, we need to specify a file name. The extension of the file name decides the kind of output file that will be created by the RMarkdown. If the extension is PDF, the output file will be a PDF file. If the extension is doc, the output file will be a Word document. If the extension is HTML, the output file will be an HTML file.

When the download button is clicked, it opens the file explorer and the file explorer asks for a file name and the directory where the output file would need to be stored. By default, the file name is as provided to the `filename` parameter. The chosen file name with its full path is the parameter passed to the function for the `content` parameter.

The parameter `content` takes a function. The main aspect to look for in this function is the call to the `rmarkdown::render()` function.

`rmarkdown::render()` creates output from the RMarkdown. The first parameter to this function is the name of the RMarkdown file to be used for creating the output from the RMarkdown. The second parameter is the `output_file` parameter that we just discussed earlier.

The third parameter is the parameters to be passed to the RMarkdown. This needs a bit of discussion. As we have already input the file in the Shiny Application and computed the word frequency, we need not do this once again in the RMarkdown. We can just pass these as parameters to the RMarkdown. So, we pass three parameters – the file name of the file to be processed for creating the word cloud, the number of words to include in the word cloud, and the data frame containing the word frequencies. The names of the parameters should be as available in the RMarkdown. So, we must have three parameters in the RMarkdown with the names and

Notice that the variable `v_params` is a list. A list is different from a vector in the sense that a list can be of different types of variables. Notice that to get the file name, we need to use the `name` variable of `input$v_input_file_txt` as

## Programming the RMarkdown

We need to make some changes to the RMarkdown as discussed earlier. The main change is that now we need to accept the parameters coming from the Shiny application. This can be done by adding the params parameter in the YAML header as shown in the following code:

```
---
```

```
title: "R Markdown Demonstration"
author: "Partha Majumdar"
date: "r format(Sys.Date(), "%B %d, %Y)"
output:
pdf_document:
fig_caption: yes
keep_tex: yes
number_sections: yes
toc: yes
html_document: default
word_document: default
params:
p_input_file: NA
p_word_frequency: NA
p_number_of_words_to_display: NA
---
```

By passing the three parameters, we have created three variables to accept the parameters – Everything else remains the same in the RMarkdown except for the fact that we do not read a file or compute the word frequency. We replace these actions by reading the values in the parameters passed to the RMarkdown. To read the values of the parameters, we need to reference the parameters as for example:

```
# Document Analysis
```

```
**File Name: 'r params$p_input_file'**
```

The complete code for the RMarkdown is provided in the next section.

## Complete code of ShabdMegh

Provided below is the complete code of ShandhMegh. The code is also available at

## app.R

```
library(shiny)
library(shinythemes)
library(shinyjs)
library(dplyr)
library(pdftools)
library(tm)
library(wordcloud)
library(DT)
library(readr)
library(textclean)
library(SnowballC)

C_SLIDER_DEFAULT_VALUE <- 50
C_TEXT_FILE_INDICATOR <- "TXT"
C_PDF_FILE_INDICATOR <- "PDF"

v_df <- reactive({
  data.frame(ind = c("test1", "test2"), n = c(10, 20))
})

v_prev_file_type <- C_TEXT_FILE_INDICATOR

# Define UI for application
ui <- fluidPage(
  shinyjs::useShinyjs(),
```

```
theme = shinytheme("united"),
tags$head(
tags$style(
HTML(".shiny-notification {

position:fixed;
top: calc(50%);
left: calc(50%);
font-family:Verdana;
fontSize:xx-large;
}
"
)
),
# Application title
fluidRow(
column(width = 3,
img(src="ParthalnPetra.jpg", width=200, height=112)
),
column(width = 6,
h1("ShabdMegh", style="color:blue; text-align: center;"),
h3("Word Cloud Producer", style="color:darkred; text-align: center;"),
),
column(width = 3,
h4("...", style="color:blue; text-align: right;"),
h4("Developed by: Partha Majumdar", style="color:blue; text-align: right;"),
h5("Riyadh (Saudi Arabia), 30-December-2019", style="color:black; text-align: right;")
)
```

```
),

fluidRow(
  column(width = 4,
  wellPanel(
    radioButtons("v_file_type", h3("Select the File Type to Upload"),
    c("Text File" = C_TEXT_FILE_INDICATOR,
    "PDF File" = C_PDF_FILE_INDICATOR),
    inline = TRUE),
  fluidRow(
    h3("Select a file from your Computer to get started to analyse the
    contents of the file.", style="color:blue; text-align: center;"),
    h4("Please select a FILE, before clicking on any TAB. This is a
    bug in the system.", style="color:darkred; text-align: center;"),
    fileInput("v_input_file_txt", h3("Choose Text File"), multiple =
    FALSE, placeholder = "No file selected", accept = c("text/csv",
    "text/comma-separated-values,text/plain", ".txt"))
  ),
  fileInput("v_input_file_pdf", h3("Choose PDF File"), multiple =
  FALSE, placeholder = "No file selected", accept =
  c("application/PDF", ".pdf"))
),
fluidRow(
  wellPanel(
    p("To learn how to use SaabdhMegh, visit the ",
    a("SaabdhMegh Demonstration Video.", href =
    "https://youtu.be/iAxIHqfEZsc", target="_blank")
  )
)
```

```
)  
,  
fluidRow(  
downloadButton("report", "Download Report")  
)  
)  
,  
column(width = 8,  
fluidRow(  
div(id = "DummyDisplay",  
img(src="CourseLogo.jpg", width=1200, height=900)  
,  
div(id = "OutputTabsPDF",  
tabsetPanel(id = "mainTabsetPDF", type = "tabs", tabPanel("DATA",  
h3("Word Frequency"),  
dataTableOutput("PDFcontents")  
,  
tabPanel("Word Cloud",  
sliderInput("v_number_of_words_pdf",  
h4("Set Number of Words to Display"),  
min = 20, max = 500,  
value = C_SLIDER_DEFAULT_VALUE,  
step = 10, ticks = TRUE),  
plotOutput("wordCloudPDF", width = "100%")  
)  
)  
,
```

```
div(id = "OutputTabsTXT",  
tabsetPanel(id = "mainTabsetTXT", type = "tabs",  
tabPanel("DATA",  
h3("Word Frequency"),
```

```

dataTableOutput("TXTcontents")
),
tabPanel("Word Cloud",
sliderInput("v_number_of_words_txt",
h4("Set Number of Words to Display"),
min = 20, max = 500,
value = C_SLIDER_DEFAULT_VALUE,
step = 10, ticks = TRUE),
plotOutput("wordCloudTXT", width = "100%")
)
)
)
)
)
)
)
)
)
)
)
)
)
)
```

```

# Define server logic required to draw a histogram
server <- function(input, output, session) {
observe({
if (input$v_file_type == C_TEXT_FILE_INDICATOR) {
shinyjs::hide("v_input_file_pdf")
shinyjs::show("v_input_file_txt")
} else {
shinyjs::show("v_input_file_pdf")
shinyjs::hide("v_input_file_txt")
}
if ((is.null(input$v_input_file_pdf) || input$v_input_file_pdf == ""))
&&
(is.null(input$v_input_file_txt) || input$v_input_file_txt == "")
})
```

```
{  
shinyjs::show("DummyDisplay")  
shinyjs::hide("OutputTabsPDF")  
shinyjs::hide("OutputTabsTXT")  
shinyjs::disable("report")  
} else {  
if (input$v_file_type == C_TEXT_FILE_INDICATOR) {  
if (is.null(input$v_input_file_txt) || input$v_input_file_txt == "")  
{  
shinyjs::hide("OutputTabsTXT")  
shinyjs::show("DummyDisplay")  
shinyjs::disable("report")  
} else {  
if (v_prev_file_type == C_PDF_FILE_INDICATOR) {  
shinyjs::hide("OutputTabsTXT")  
shinyjs::show("DummyDisplay")  
shinyjs::disable("report")  
} else {  
shinyjs::show("OutputTabsTXT")  
shinyjs::hide("DummyDisplay")  
shinyjs::enable("report")  
}  
}  
}  
}  
shinyjs::hide("OutputTabsPDF")  
v_prev_file_type <- C_TEXT_FILE_INDICATOR  
} else {  
if (is.null(input$v_input_file_pdf) || input$v_input_file_pdf == "")  
{  
shinyjs::hide("OutputTabsPDF")  
shinyjs::show("DummyDisplay")  
shinyjs::disable("report")  
}
```

```

} else {
if (v_prev_file_type == C_TEXT_FILE_INDICATOR) {
shinyjs::hide("OutputTabsPDF")
shinyjs::show("DummyDisplay")
shinyjs::disable("report")
} else {
shinyjs::show("OutputTabsPDF")
shinyjs::hide("DummyDisplay")
shinyjs::enable("report")
}
}
shinyjs::hide("OutputTabsTXT")
v_prev_file_type <<- C_PDF_FILE_INDICATOR
}
}
})
observeEvent(input$v_input_file_pdf, {
# Reset Display to DATA TAB
updateTabsetPanel(session, "mainTabsetPDF",
selected = "DATA"
)
# Reset the Slider
updateSliderInput(session, 'v_number_of_words_pdf', value =
C_SLIDER_DEFAULT_VALUE)
})
observeEvent(input$v_input_file_txt, {
# Reset Display to DATA TAB
updateTabsetPanel(session, "mainTabsetTXT",
selected = "DATA"
)
# Reset the Slider
}

```

```
updateSliderInput(session, 'v_number_of_words_txt', value =
C_SLIDER_DEFAULT_VALUE)
})
output$PDFcontents <- renderDataTable({
req(input$v_input_file_pdf)
v_df <- reactive({
# Read the PDF File
v_raw_text <- pdf_text(input$v_input_file_pdf$datapath) %>%
strsplit("\n")
# Calculate Word Frequency
calculateWordFrequency(v_raw_text)
})
displayDataTable()
})
output$TXTcontents <- renderDataTable({
req(input$v_input_file_txt)

v_df <- reactive({
# Read the TXT File
v_text <- tibble(text = gsub("[^[:alnum:]]//'", "", 
read_lines(input$v_input_file_txt$datapath)))
# Drop the Empty Lines
v_raw_text <- drop_empty_row(v_text)
# Calculate Word Frequency
calculateWordFrequency(v_raw_text)
})
displayDataTable()
})
displayDataTable <- function() {
DT::datatable(
{v_df()},
caption = htmltools::tags$caption(
```

```

style = 'caption-side: bottom; text-align: center;',
'Table 1: ', htmltools::em('Word Frequency.')
),
extensions = 'Buttons',
options = list(
fixedColumns = TRUE,
autoWidth = TRUE,
ordering = TRUE,
dom = 'Bftsp',
buttons = c('copy', 'csv', 'excel')
))
}

calculateWordFrequency <- function(l_text) {
v_data <- reactive({
# Create the Corpus
v_corpus <- Corpus(VectorSource(l_text))
# Create the Term Document Matrix after cleaning the Corpus
v_tdm <- TermDocumentMatrix(v_corpus,
control =
list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf)))
)
)
# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)
# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

```

```

# Sort the Words in DESCENDING Order and create Data Frame
stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))
})
return (v_data())
}

##### WORD CLOUD
output$wordCloudPDF <- renderPlot({
req(input$v_input_file_pdf)
generateWordCloud(input$v_number_of_words_pdf)
})

##### WORD CLOUD
output$wordCloudTXT <- renderPlot({
req(input$v_input_file_txt)
generateWordCloud(input$v_number_of_words_txt)
})

generateWordCloud <- function(l_number_of_words) {
withProgress(message = 'Generating',
detail = 'This may take a while...', value = 0, {
wordcloud(words = v_df()$ind, freq = v_df()$values, min.freq = 1,
max.words = l_number_of_words,
random.order=FALSE,
colors=brewer.pal(8, "Dark2")
)
})
}
output$report <- downloadHandler(
# For PDF output, change this to "report.pdf"
filename <- "report.pdf",
content <- function(file) {
withProgress(message = 'Generating Report. This takes a minute
or two...', value = 0, {

```

```
if (input$v_file_type == C_TEXT_FILE_INDICATOR) {  
  l_file_name <- input$v_input_file_txt$name  
  l_words_to_display <- input$v_number_of_words_txt  
} else {  
  l_file_name <- input$v_input_file_pdf$name  
  l_words_to_display <- input$v_number_of_words_pdf  
}  
  
# Set up parameters to pass to Rmd document  
v_params <- list(  
  p_input_file = l_file_name,  
  p_number_of_words_to_display = l_words_to_display,  
  p_word_frequency = v_df()  
)  
rmarkdown::render("ShabdhMegh.Rmd",  
  output_file = file,  
  params = v_params,  
  envir = new.env(parent = globalenv())  
)  
}  
}  
}  
}  
}  
}  
}  
}  
}  
# Run the application  
shinyApp(ui = ui, server = server)
```

## ShabdhMegh.RMD

```
---
```

```
title: "R Markdown Demonstration"
author: "Partha Majumdar"
date: "r format(Sys.Date(), "%B %d, %Y")"
output:
pdf_document:
fig_caption: yes
keep_tex: yes
number_sections: yes
toc: yes
html_document: default
word_document: default
params:
p_input_file: NA
p_word_frequency: NA
p_number_of_words_to_display: NA
---
'''{r setup, include=FALSE, echo=FALSE, warning=FALSE}
knitr::opts_chunk$set(echo = TRUE)
'''
'''{r load_libraries, include=FALSE, echo=FALSE, warning=FALSE}
library(wordcloud)
'''
# Document Analysis
**File Name: 'r params$p_input_file'**
```

The document provides basic analysis of the file \*'r params\$p\_input\_file'\*.

This document is developed to demonstrate generating \*\*R Markdown Documents\*\*.

\* This program does the following:

+ Load the needed libraries

+ Read the File

+ Compute the Word Frequency after cleaning the data in the file

+ Form the Word Cloud from the Word Frequency provided as an Input Parameter

```
'''{r gather_word_frequency, include=FALSE, echo=FALSE,
warning=FALSE}
```

```
v_df <- params$p_word_frequency
```

```
'''
```

```
## Most Frequent Words
```

There are 'r nrow(v\_df)' unique words in the Document.

Given below are the most frequent words used in the Document.

```
'''{r most_frequent_words, include=TRUE, echo=TRUE,
warning=FALSE}
```

```
head(v_df)
```

```
'''
```

```
## Word Cloud
```

Below is the Word Cloud of the most frequent words used in the Document. The Size and the Color of the Words in the Word Cloud is set based on the frequency of the word.

```
### Generating displaying the Code and Suppressing the Warnings
```

The code used to generate the Word Cloud has been displayed in the document. This can be accomplished by setting the parameter **\*\*echo\*\*** equal to TRUE.

The **\*warnings\*** can be suppressed by setting the parameter **\*\*warning\*\*** to FALSE.

```
'''{r generate_word_cloud, include=TRUE, echo=TRUE,
warning=FALSE}
wordcloud(words = v_df$ind, freq = v_df$values, min.freq = 1,
max.words=params$p_number_of_words_to_display,
colors=brewer.pal(8, "Dark2"),
random.order = FALSE)
'''
```

### Generating Word Cloud NOT displaying the Code, but  
SHOWING the Warnings

```
'''{r generate_word_cloud_2, include=TRUE, echo=FALSE,
warning=TRUE}
wordcloud(words = v_df$ind, freq = v_df$values, min.freq = 1,
max.words=params$p_number_of_words_to_display,
colors=brewer.pal(8, "Dark2"),
random.order = FALSE)
'''
\center
style="color:red">**----- THE END -----**\n\center
```

## Conclusion

Through a simple application, we have discussed many important aspects of Shiny application programming. We came across many new libraries available in R while discussing the different code snippets to achieve these results.

We learned about a whole new dimension of R programming when we discussed RMarkdown. We discussed how to format text using RMarkdown. We learned how to embed R code in RMarkdown. We found out about the different parameters using which we can control the output from the R code chunks. And lastly, we explored how to use Latex code in RMarkdown.

Then we discussed the new aspects of Shiny programming including creating Tabs and radio buttons, inputting files, using JavaScript in Shiny application using shinyjs library, etc. On the server-side Shiny programming, we learned how to make the code responsive using the observe() function and the reactive() function. We also discussed how to produce output in a Data Table using the DT library. Lastly, we discussed how we to integrate Shiny applications and RMarkdown.

In the remainder of the book, we will discuss the core contents of the book - **Emotion** We will apply all the concepts learned so far to create a product for emotion analysis in the remaining chapters.

## Points to Remember

RMarkdown can be used to create dynamic documents which can include R code.

RMarkdown can be used to produce documents in HTML, PDF, and Word document formats.

Tabset panel can be created in Shiny applications using the tabsetPanel() function. Inside a Tabset Panel, we can create tabs in Shiny applications using the function tabPanel().

Radio Buttons can be created in Shiny applications using the radioButtons() function.

To input files in Shiny applications, we use the function

To upload PDF files, we can use the function pdf\_text() from the library

shinyjs::useShinyjs() needs to be the first line of code in the ui() function for the JavaScript code to work in a Shiny Application.

shinyjs::hide() and shinyjs::show() functions are used to hide or show a control, respectively.

To update a global variable, we must use the operator

Functions `dataTableOutput()` on the client-side and `renderDataTable()` on the server-side are used to produce data table output.

The `withProgress()` function can be used to produce the progress bar.

The `downloadButton()` function in the client-side creates the `button.download`

The `downloadHandler()` function needs to be programmed on the server-side for creating downloads from a Shiny Application.

The function `rmarkdown::render()` generates the output from an RMarkdown.

### Multiple choice questions

YAML stands for:

Yet Another Markup Language

Yet Another Manipulation Language

YAML Ain't Markup Language

None of these

Using RMarkdown, it is NOT possible to create output in which of these formats?

PDF

XSL

HTML

DOC

To use JavaScript in Shiny Applications, we need to load which library?

shinyjs

ShinyJS

shinyJS

None of the above

To use a data table in Shiny Applications, we need to load which library?

dataTable

DT

Data table

None of the above

To access the parameters in RMarkdown, we need to access which of the following variables?

Parameters cannot be accessed on RMarkdown

Parameters cannot be passed to RMarkdown

parameter

params

## Answers to MCQs

C

B

A

B

D

## Questions

Enhance the code of ShabdhMegh to also accept Word documents as input for generating a word cloud.

### Key terms

**YAML:** It stands for YAML Ain't Markup Language. YAML is a human-friendly data serialization standard for all programming languages. It is commonly used for the configuration of files and in applications where data is being stored and transmitted.

**PDF:** It stands for Portable Document Format. PDF files are used for presenting and exchanging documents reliably and are independent of software, hardware, and operating systems.

### SECTION - 3

## CHAPTER 7

### Sentiment Analysis

So far in this book, we discussed the essentials of R programming and Shiny programming. We focused on the topics in R programming and Shiny programming that we need for developing data products for emotion analysis.

From this chapter onwards, we start our exploration into ***emotion***. We start this journey with ***sentiment***. We will discuss the meaning and the applications of sentiment analysis. We will also learn about the different approaches to sentiment analysis. In this chapter, we will explore the different building blocks available in R Language which can help develop systems for sentiment analysis and emotion analysis. We will discuss key components, like Lexicons, which are essential for developing a system for sentiment analysis and emotion analysis.

In this chapter, we will develop a system using R programming which will be able to distinguish between positive sentiments and negative sentiments in a given text.

## Structure

In this chapter, we will discuss the following topics:

What is sentiment analysis?

Sentiment knowledge bases in R

Conducting sentiment analysis

- Extracting words from text
- Generating word frequency
- Finding sentiment across the text
- Determine the contribution of each word in the sentiment score
- Visualizing words with positive and negative sentiment

## Objectives

After studying this unit, you should be able to:

Understand the concept of sentiment analysis

Understand the Lexicons available in R language for sentiment analysis

Conduct sentiment analysis for a text

## What is sentiment analysis?

Sentiment **Analysis** is also known as **Opinion Mining** or **Emotion AI**. Sentiment Analysis is a branch of **Natural Language Processing**. In Sentiment Analysis, we conduct Text Analysis by applying Computational Linguistic to systematically identify, extract, quantify, and study the effective states of expressions in the text in any language. Sentiment Analysis is widely used in analyzing the voice of the customers through reviews and surveys. Sentiment Analysis is widely used in analyzing online media, especially social media. It is widely used in marketing exercises and in clinical medicine.

In sentiment analysis, we will try to determine the polarity of a given text. We will figure out if the text has a positive tone or a negative tone or a neutral tone. We can extend sentiment analysis to determine emotions in a text. We can classify whether the text expresses a happy or a sad emotion, or an emotion of fear or anticipation, etc. An application of this could thought of as to analyze analyzes an email and figure out whether the person has written the email in a happy mood or an angry mood, etc.

We will use emotion analysis in our workplace for two applications. We will analyze all the emails exchanged with our customers and vendors to try to figure out the sentiments and/or emotions in the conversation. Where we see a concern, we plan corrective measures.

Another area where we apply sentiment/emotion analysis is in analyzing the resumes of the prospective candidates who apply to our organization. This analysis helps us filter out just the candidates we would like to interview from a huge pile of applications. We can use speech to text technology to create text for all spoken sentences during a conversation. On this text, we can apply sentiment analysis. Then, we would be able to figure out the sentiments and/or emotions expressed during any conversation.

So, the applications of sentiment analysis are many. Several more possible applications sentiment analysis is evolving every day.

## Approaches to sentiment analysis

There are three approaches to sentiment analysis.

## Knowledge-based approach

In knowledge-based approaches, sentiment is seen as the function of keywords (usually based on their count). Thus, the main task is the construction of Sentiment discriminatory word lexicons with indicated class labels (positive or negative or neutral) and sometimes even with their intensiveness. The keystone of the knowledge-based approach is the lexicon that is applied to recognize sentiment words in the text.

Knowledge-based approaches for sentiment analysis are also called systems for sentiment analysis. In this approach, we analyze the text based on a fixed set of rules. Usually, a rule-based system uses a set of human-crafted rules to help identify subjectivity, polarity, or the subject of an opinion.

In this book, we will use knowledge-based approach to extract emotions from text.

### Statistical approach

The statistical approach to sentiment analysis is a **term counting** term counting method used to classify text as positive, negative, or neutral. In this method, the Lexicon is applied by counting positive and negative words found in a text and determining the sentiment polarity based on which a class received the highest score.

The Statistical approach for Sentiment Analysis relies on machine learning techniques to learn from data. In this technique, a sentiment analysis task is modeled as a classification problem, whereby a classifier is fed a text and it returns a category, for example, positive, negative, or neutral.

It needs to be noted that solving a classification problem classification problem needs a knowledge base to be created by humans. The knowledge base that is created for sentiment analysis is called the Lexicon. Based on this knowledge base, the machine is trained. We call this form of problem solving supervised learning.

## Hybrid approach

The hybrid approach of sentiment analysis exploits both statistical methods and knowledge-based methods for polarity detection. It inherits high accuracy from machine learning (statistical machine learning (statistical methods) and stability from the lexicon-based approach.

**Difference between traditional programming and machine learning.**

**In traditional programming, we input the data & the rules and the program generates the output.**



**Figure 7.1: Traditional programming**

**In machine learning, we input the data & the output and the program generates the rules.**



**Figure 7.2: Machine learning**



## Sentiment knowledge bases in R

As I stated, we will use knowledge-based approach to sentiment analysis in this book. In R Language, knowledge bases for sentiment analysis are available in the library called We can load the tidytext library as shown below:

```
library(tidytext)
```

In the tidytext library, there is a data set called the sentiments. The sentiments data set has 6,786 words classified as emoting positive or negative sentiments. We can see the sentiments data set as shown in the following code:

```
sentiments
## # A tibble: 6,786 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 2-faces    negative
## 2 abnormal    negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate   negative
## 7 abomination negative
## 8 abort       negative
## 9 aborted     negative
```

```
## 10 aborts      negative  
## # ... with 6,776 more rows
```

You can see that the sentiments data set is a data frame with two columns – word and sentiment. We can see the words with a positive sentiment in the sentiments data frame as shown in the following code:

```
subset(sentiments, sentiment == 'positive')  
## # A tibble: 2,005 x 2  
##   word      sentiment  
##   <chr>     <chr>  
## 1 abound    positive  
## 2 abounds   positive  
## 3 abundance positive  
## 4 abundant  positive  
## 5 accessable positive  
## 6 accessible positive  
## 7 acclaim    positive  
## 8 acclaimed  positive  
## 9 acclamation positive  
## 10 accolade  positive  
## # ... with 1,995 more rows
```

We can see that there are 2,005 words with positive **sentiments** in the sentiments data frame. Similarly, we can see the words with a negative sentiment in the sentiments data frame using the command as shown in the following code. I have not included the output of the command and left it for you to check:

```
subset(sentiments, sentiment == 'negative')
```

These data sets with words classified by the sentiments are called lexicons. There are various lexicons in R. We will discuss the popular lexicons available in R.

## [afinn Lexicon](#)

afinn lexicon has words classified as emoting positive or negative or neutral sentiments. In this lexicon, each word is given a value between -5 and 5. Words with a negative value are words with a negative sentiment. The higher the negative value of the word, greater is the negative sentiment that the word emotes. Similarly, words with a positive sentiment have a positive value; the higher the positive value, greater is the positive sentiment that the word emotes. The words with a value of 0 are words which emote a neutral sentiment.

We can see the lexicon using the function `get_sentiments()` from the `tidytext` library as shown in the following code:

```
get_sentiments("afinn")
## # A tibble: 2,477 x 2
##   word      value
## 
## 1 abandon    -2
## 2 abandoned  -2
## 3 abandons   -2
## 4 abducted   -2
## 5 abduction  -2
## 6 abductions -2
## 7 abhor      -3
## 8 abhorred   -3
## 9 abhorrent  -3
```

```
## 10 abhors      -3  
## # ... with 2,467 more rows
```

We can see that there are 2,477 words in the **afinn**. We can see the subsets of the **afinn lexicon** as shown in the following code. The following command shows the words in the **afinn lexicon** with a value of 0:

```
subset(get_sentiments("afinn"), value==0)  
## # A tibble: 1 x 2  
##   word     value  
##  
## 1 some kind     0
```

The following command shows the words in the **afinn lexicon** with a value of 2:

```
subset(get_sentiments("afinn"), value==2)  
## # A tibble: 448 x 2  
##   word     value  
##  
## 1 abilities     2  
## 2 ability       2  
## 3 absolve       2  
## 4 absolved      2  
## 5 absolves      2  
## 6 absolving     2  
## 7 accomplish    2  
## 8 accomplished   2  
## 9 accomplishes   2
```

```
## 10 acquit          2  
## # ... with 438 more rows
```

The following command shows the words in the afinn lexicon with a value of -4:

```
subset(get_sentiments("afinn"), value===-4)  
## # A tibble: 43 x 2  
##   word      value  
##  
##   1 ass        -4  
##   2 assfucking -4  
##   3 asshole    -4  
##   4 bullshit    -4  
##   5 catastrophic -4  
##   6 damn       -4  
##   7 damned     -4  
##   8 damnit     -4  
##   9 dick        -4  
##  10 dickhead   -4  
## # ... with 33 more rows
```

## bing Lexicon

The **bing lexicon** is the same as the sentiments data set. We can view the bing lexicon as shown in the following code:

```
get_sentiments("bing")
## # A tibble: 6,786 x 2
##   word      sentiment
## 
## 1 2-faces    negative
## 2 abnormal    negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate   negative
## 7 abomination negative
## 8 abort       negative
## 9 aborted     negative
## 10 aborts     negative
## # ... with 6,776 more rows
```

## **loughran Lexicon**

The loughran lexicon was created for text analysis of financial documents by *Tim Loughran* and *Bill McDonald* in 2011. The loughran lexicon has 4,150 words classified in **six** sentiments.

We can view the loughran lexicon as shown in the following code:

```
get_sentiments("loughran")
## # A tibble: 4,150 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 abandon    negative
## 2 abandoned  negative
## 3 abandoning negative
## 4 abandonment negative
## 5 abandonments negative
## 6 abandons   negative
## 7 abdicated  negative
## 8 abdicates  negative
## 9 abdicating negative
## 10 abdication negative
## # ... with 4,140 more rows
```

We can view sentiments in which the words are classified in the **loughran lexicon** as shown in the following code:

```
unique(get_sentiments("loughran")$sentiment)
## [1] "negative"      "positive"
"uncertainty"    "litigious"     "constraining"
## [6] "superfluous"
```

I leave it for you to check the words with different sentiments in the loughran lexicon.

## nrc Lexicon

The last lexicon we will discuss is the nrc lexicon. The nrc lexicon was introduced to the world by *Saif Mohammad* and *Peter Turney* on May, 2010. The nrc lexicon can be used for Emotion Analysis and has emotion analysis. 13,901 words classified in ten sentiments.

We can view the **nrc lexicon** as shown in the following code:

```
get_sentiments("nrc")
## # A tibble: 13,901 x 2
##       word      sentiment
##   <chr>     <chr>
## 1 abacus    trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
## 6 abandoned fear
## 7 abandoned negative
## 8 abandoned sadness
## 9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

We can view sentiments in which the words are classified in the **nrc lexicon** as shown in the following code:

```
unique(get_sentiments("nrc")$sentiment)
## [1] "trust"          "fear"           "negative"
"sadness"         "anger"

## [6] "surprise"       "positive"
"disgust"         "joy"            "anticipation"
```

The **nrc lexicon** classifies the words in eight emotions<sup>8</sup> emotions (trust, surprise, joy, anticipation, fear, sadness, anger, disgust) and two sentiments<sup>2</sup> sentiments (positive, negative). The classification of trust, surprise, joy, and positive are considered as positive sentiment. While fear, sadness, anger, and negative are considered as negative sentiment.

Each word in the nrc lexicon may be associated with multiple sentiments.

We will discuss writing a program using R language for emotion analysis with the help of the nrc lexicon in [Chapter 8: Emotion](#)

nrc Lexicon can be used for non-commercial research work free of cost. However, for using the nrc Lexicon for commercial use, one needs purchasing appropriate licenses. The implication is that when we discuss the Shiny applications, we will develop for emotion analysis using the nrc Lexicon in [Chapter 12:](#) we need noting that to publish those Shiny applications on World Wide Web (WWW), we need acquiring the license for nrc Lexicon.

**Without the license, the Shiny Applications will function properly  
on the Local Machines; but not when published on the Internet.  
You can read about nrc Lexicon at**

## Conducting sentiment analysis

Now that we have discussed the building blocks for sentiment analysis, let us discuss the process of sentiment analysis. The first step is to gather data on which we would conduct sentiment analysis. With regards to data, we can consider any text. However, so that have a reasonably large data set to analyze, we will consider books for the source of data.

We can get the complete works of Shakespeare from the Project Gutenberg website. We can download the data as shown in the following code. The data is downloaded from the website and stored in a file named

```
TEXTFILE = "./shakespeare.txt"  
download.file("http://www.gutenberg.org/cache/epub/100/pg100.txt",  
destfile = TEXTFILE)
```

The function `download.file()` can be used to download a file from a website. The first parameter is the location of the file on the internet and the second parameter – `destfile` - is the file where the contents will be saved.

Once the data is downloaded, the file can be read using the function `readLines()`. We need to provide the complete file path (with the file name) to the function. The code for reading the file is shown in the following code:

```
v_shakespeare = readLines(TEXTFILE)
head(v_shakespeare)
## [1] "The Project Gutenberg EBook of The Complete Works of
William Shakespeare, by"
## [2] "William Shakespeare"
## [3] ""
## [4] "This eBook is for the use of anyone anywhere at no cost
and with"
## [5] "almost no restrictions whatsoever. You may copy it, give it
away or"
## [6] "re-use it under the terms of the Project Gutenberg
License included"
```

You can see above that the `readLines()` function returns a set of vectors for the contents of the file. We need to convert these vectors to a data frame using the `data.frame()` function before we can use this data for analysis. The code is as shown in the following code:

```
v_text <- data.frame(text = v_shakespeare)
head(v_text)
##      text
## 1 The Project Gutenberg EBook of The Complete Works of
William Shakespeare, by
## 2      William Shakespeare
## 3
## 4 This eBook is for the use of anyone anywhere at no cost
and with
```

```
## 5      almost no restrictions whatsoever. You may copy it,  
give it away or  
## 6      re-use it under the terms of the Project  
Gutenberg License included
```

Use the `nrow()` function on the data frame `v_text` to find that there are 1,24,787 rows in the data frame.

Now that we have the data, we can start our analysis.

## Extracting words from the text

The first step is to extract words from the text. This can be done using the following code. Please note that there are many other ways to achieve the same result in R:

```
v_tidy_data <- v_text %>%
  mutate(linenumber = row_number()) %>%
  ungroup() %>%
  unnest_tokens(word, text)
```

The `mutate()` function and the `ungroup()` function are available in the `dplyr` library. The pipe operator is available in the `dplyr` library. The function `unnest_token()` is available in the `tidytext` library.

Using the `mutate()` function, we can create a new column called `linenumber` to contain the row from which the word has been extracted. We use the `ungroup()` function as a precaution in case there are any groups in the data.

Lastly, we use the `unnest_token()` function to return each word in the text as a token. Notice that we have passed the parameter `word` to the `unnest_token()` function. This instructs the `unnest_token()` function to return each word in the text as a token. Each word in a text is also referred to as 1-gram. We will see in [Chapter 8: Emotion Analysis](#) that we can use the `unnest_token()` function to extract two consecutive words (called

bigram or 2-grams) or three consecutive words (called trigram or 3-grams).

The output of this command is as shown in the following code:

```
head(v_tidy_data, 10)
##      linenumber      word
## 1          1       the
## 1.1        1   project
## 1.2        1  gutenberg
## 1.3        1     ebook
## 1.4        1       of
## 1.5        1       the
## 1.6        1  complete
## 1.7        1     works
## 1.8        1       of
## 1.9        1  william
```

So, we have a data frame called **v\_tidy\_data** with two columns – linenumber and word.

## Generating the words frequency

The next step is to check the word frequency. We can achieve this using the code as shown in the following code:

```
positive_sentiment <- get_sentiments("bing") %>%
  filter(sentiment == "positive")
v_tidy_data %>%
  semi_join(positive_sentiment) %>%
  count(word, sort = TRUE)
## Joining, by = "word"
## # A tibble: 922 x 2
##       word        n
##   <chr>    <dbl>
## 1 good     2834
## 2 well      2241
## 3 love      2109
## 4 like      1786
## 5 great     914
## 6 sweet     801
## 7 master    777
## 8 fair      768
## 9 heaven    621
## 10 noble    615
## # ... with 912 more rows
```

The one addition that we do in this code is to generate the word frequency of the words with a positive sentiment only. For this, we will first extract the words with a positive sentiment from the **bing lexicon** using the filter() function from the dplyr library. Then, we will perform a semi join between the words extracted from the text and the words with a positive sentiment extracted from the bing Lexicon. We can perform a semi join using the function semi\_join() from the dplyr library.

We will generate the word frequency using the count() function from the dplyr library.

**When any join operation is performed, we must pass 2 data sets. We refer to the first data set as the left data set and the second data set as the right data set. Semi Join is a filtering join which returns all rows from X where there are matching values in Y, keeping just columns from X. Here X is the left data set and Y is the right data set.**

### Finding sentiment across the text

Now, we will find sentiments across the text As you notice from the work done so far, we have the words in the text along with the line number where those words exist in the text. So, we should be able to determine the sentiments for every line of the text.

We will go through the process step-by-step.

## Determine the sentiment of each word

First, we will determine the sentiment of each word using the following code. We can accomplish this by making an inner join between the words in the text and the words in the lexicon as shown in the following code:

```
v_text_sentiment <- v_tidy_data %>%
  inner_join(get_sentiments("bing"))
## Joining, by = "word"
```

The `inner_join()` function is available in the `dplyr` library. The Pipe is available in the `dplyr` library. The common column between two sets of data is `word`. So, the join is performed using the column `word`.

**When any join operation is performed, we must pass 2 data sets. We refer to the first data set as the left data set and the second data set as the right data set. Inner join is a mutating join which return all rows from X where there are matching values in Y.**

The output is as shown in the following code:

```
head(v_text_sentiment, 5)
##   linenumber      word sentiment
## 1          1    works   positive
## 2          12   works   positive
```

```
## 3      22      works  positive
## 4      41      works  positive
## 5      44      readable  positive
```

### Determine the sentiment of each line

Next, we will find sentiments in each line. We extend the code in the previous section by aggregating the sentiments using the count() function from the dplyr library:

```
v_text_sentiment <- v_tidy_data %>%
  inner_join(get_sentiments("bing")) %>%
  count(index = linenumbers %% 80, sentiment)
## Joining, by = "word"
```

Notice that we have created a new column called index. I have created chunks of 80 lines. We have seen that there are 1,24,787 lines of text in the data that we are analyzing So, if we take the counts of each line, we will have a huge data frame. So, I have shortened the output data frame by a factor of 80.

**The %/ operator from dplyr library returns the quotient from a division. So,**

**19 %/ 80 = 0  
89 %/ 80 = 1  
229 %/ 80 = 2  
and so on.**

The output of the above code is as shown below. We have the count of words with positive and negative sentiments in each

chunk of the 80 lines:

```
head(v_text_sentiment, 15)
## # A tibble: 15 x 3
##   index sentiment     n
##
##   1     0 negative     1
##   2     0 positive    12
##   3     1 negative   29
##   4     1 positive   15
##   5     2 negative   21
##   6     2 positive   27
##   7     3 negative   20
##   8     3 positive   37
##   9     4 negative   30
##  10     4 positive   31
##  11     5 negative   16
##  12     5 positive   31
##  13     6 negative   25
##  14     6 positive   34
##  15     7 negative   22
```

### Separate the counts of positive and negative sentiments

In the preceding output data frame, we saw that we have the counts of positive and negative sentiments for each chunk of the 80 lines. Now, we will create separate columns for the counts of positive and negative sentiments in each chunk.

Notice that words with positive or negative sentiments are available in the column sentiment. We use the spread() function in the tidyverse library to accomplish this. The spread() function, when applied on a column, creates a separate column for each unique value in that column. We use the following code as shown:

```
library(tidyverse)
```

```
v_text_sentiment <- v_tidy_data %>%
  inner_join(get_sentiments("bing")) %>%
  count(index = linenumbers %/ 80, sentiment) %>%
  spread(sentiment, n, fill = 0)
## Joining, by = "word"
```

```
head(v_text_sentiment, 20)
## # A tibble: 20 x 3
##       index negative positive
##       <dbl>     <dbl>    <dbl>
## 1         0        0       12
## 2         1        1       29
## 3         2        1       15
## 4         3        0       10
## 5         4        0       10
## 6         5        0       10
## # ... with 14 more rows
```

```
## 3 2 21 27
## 4 3 20 37
## 5 4 30 31
## 6 5 16 31
## 7 6 25 34

## 8 7 22 39
## 9 8 38 38
## 10 9 37 36
## 11 10 24 46
## 12 11 27 32
## 13 12 31 25
## 14 13 16 35
## 15 14 27 26
## 16 15 34 31
## 17 16 33 37
## 18 17 33 32
## 19 18 18 33
## 20 19 20 38
```

The second parameter of the `spread()` function is the column whose value has to be assigned to each row of the new columns. The `fill` parameter is the default value to assign in case no entry for a column is found for a row.

## Calculate the net sentiments score for each chunk

Now that we have the counts of positive sentiments and negative sentiments in each chunk of the 80 lines, we can compute the Net Sentiment Score for each chunk. We find the net sentiment score by subtracting the count of the negative sentiments from the count of the positive sentiments. We will create a separate column to store this value and call this column sentiment. We can create a new column in a data frame using the mutate() function from the dplyr library. The code and the output are as shown in the following code:

```
v_text_sentiment <- v_tidy_data %>%
  inner_join(get_sentiments("bing")) %>%
  count(index = linenumbers %/ 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
## Joining, by = "word"

head(v_text_sentiment, 20)
## # A tibble: 20 x 4
##       index negative positive sentiment
##       <dbl>     <dbl>     <dbl>      <dbl>
## 1         0         1        12        11
## 2         1         29        15       -14
## 3         2         21        27         6
## 4         3         20        37        17
## 5         4         30        31         1
```

##	6	5	16	31	15
##	7	6	25	34	9
##	8	7	22	39	17
##	9	8	38	38	0
##	10	9	37	36	-1
##	11	10	24	46	22
##	12	11	27	32	5
##	13	12	31	25	-6
##	14	13	16	35	19
##	15	14	27	26	-1
##	16	15	34	31	-3
##	17	16	33	37	4
##	18	17	33	32	-1
##	19	18	18	33	15
##	20	19	20	38	18

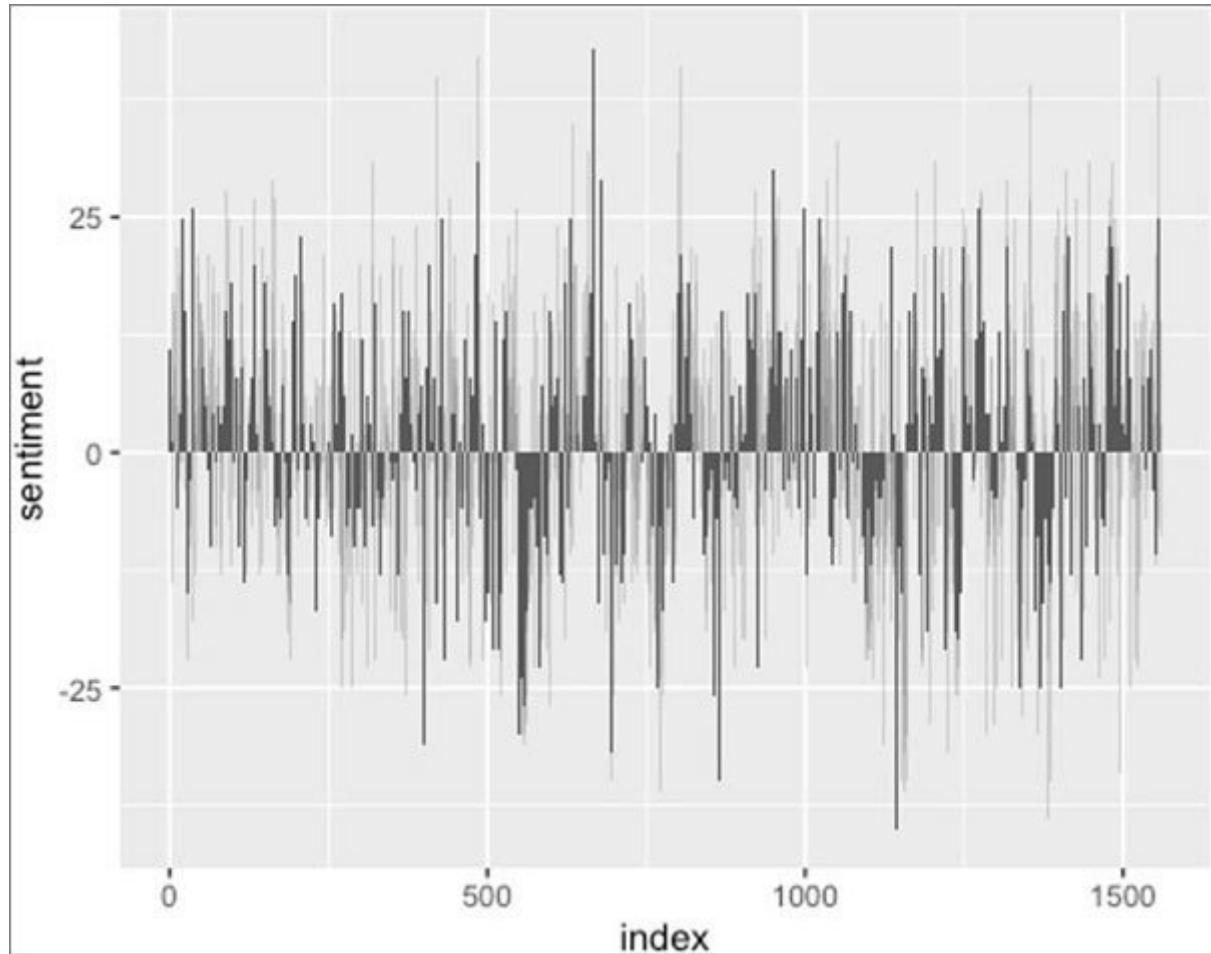
### Plot the net sentiments score

Now, we can plot the net sentiment score. The visualization will provide us an idea of how the sentiments have been expressed in the text as it progressed from the start to the end.

We use the `ggplot()` function from the `ggplot2` library to create this visualization as shown in the following code:

```
library(ggplot2)
ggplot(v_text_sentiment, aes(index, sentiment)) + geom_bar(stat =
"identity", show.legend = TRUE)
```

The output is as shown in following [figure](#)



*Figure 7.3: Progression of sentiments expressed in the text from start to end*

### Determining the contribution of each word in the sentiment score

We have determined how the sentiment expressions have progressed in the text. Using this program, given any text, we can find out the way sentiments are expressed. Like I discussed earlier, we use this analysis on text like email conversations with our partners. We can determine whether there is a sense of negativity creeping in the conversation in the email exchanges. When we sense this, we can take corrective measures.

Similarly, when a person is authoring any document, he/she can determine how the sentiments are being expressed in the document. Based on this analysis, he/she can determine whether the document is progressing the way he/she planned and/or envisaged. Suitable alterations can be made based on this analysis.

Next, we will determine how the words used in the text are contributing to the different sentiments. For this, we need to determine the sentiment score of each word in the text. We can do this using the following code as shown. You can see the output as well:

```
v_word_count <- v_tidy_data %>%
inner_join(get_sentiments("bing")) %>%
count(word, sentiment, sort = TRUE)
## Joining, by = "word"
```

```

head(v_word_count, 20)
## # A tibble: 20 x 3
##   word      sentiment     n
##
## 1 good      positive    2834
## 2 well      positive    2241
## 3 love      positive    2109
## 4 like      positive    1786
## 5 great     positive    914
## 6 death     negative    869
## 7 sweet     positive    801
## 8 master    positive    777
## 9 fair      positive    768
## 10 poor     negative    629
## 11 fear     negative    627
## 12 heaven   positive    621
## 13 noble    positive    615
## 14 better    positive    594
## 15 grace    positive    577
## 16 peace    positive    539
## 17 dead     negative    533
## 18 die      negative    475
## 19 fool     negative    465
## 20 mistress  negative    451

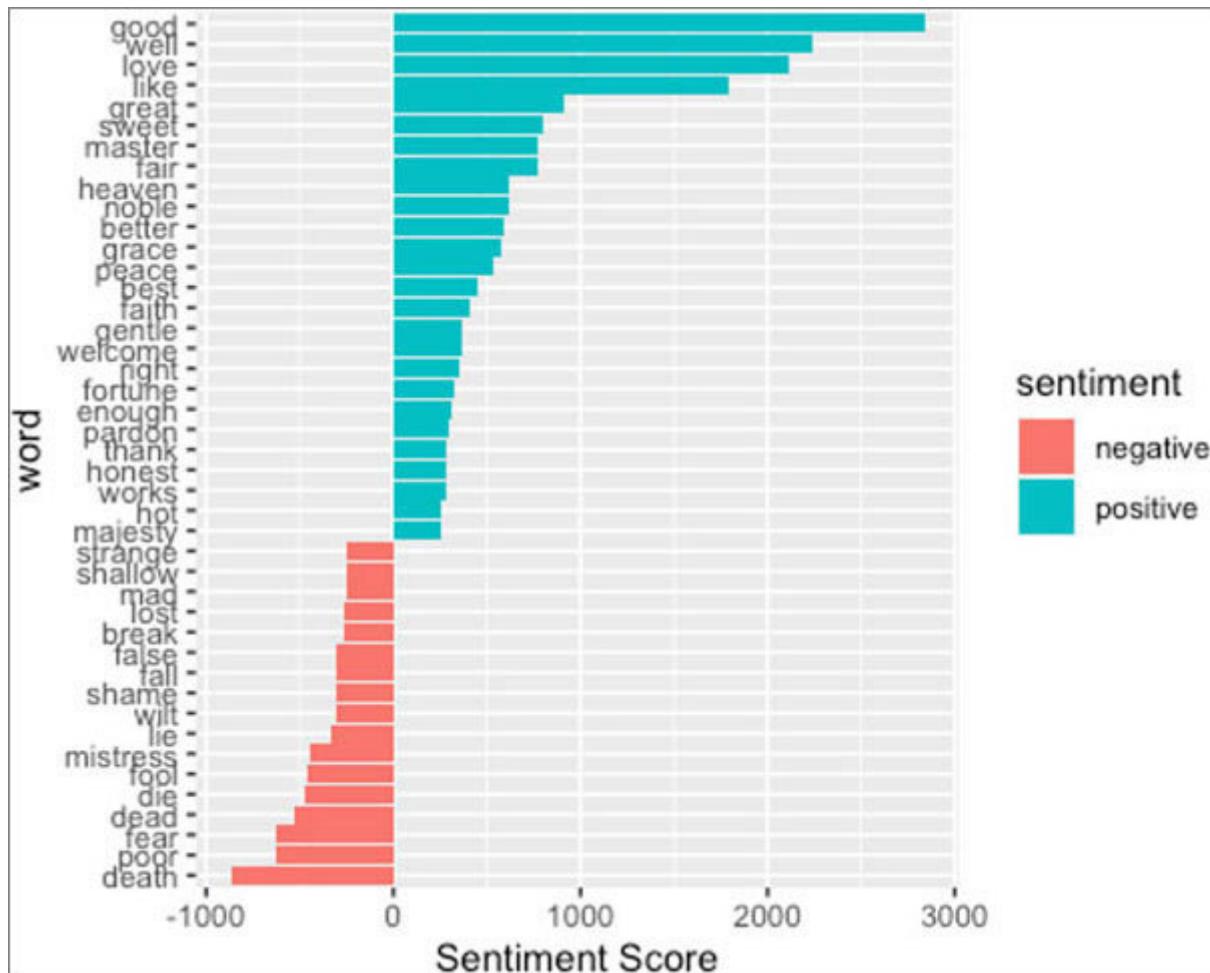
```

This code should be clear based on the previous discussions. So, we have the sentiment score for each word and the above output displays the same in the descending order. Now, we can visualize

this data using the `ggplot()` function as shown in the following code:

```
v_word_count %>%
  filter(n > 250) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col() +
  coord_flip() +
  labs(y = "Sentiment Score")
```

Notice that I have made the negative sentiment score a negative number so that the negative sentiment scores and the positive sentiment scores point to the opposite directions in the visualization. The output is as shown in the following screenshot:



*Figure 7.4: Sentiment score per word*

## Visualizing words with positive and negative sentiment

Now, we will create a word cloud of words in the text with positive sentiment and negative sentiment. This visualization can provide instant feedback about the document. For example, such visualizations can be used to analyze a resume.

**One need being careful while interpreting such visualizations. While analyzing the resume of one of my customer while trying my sentiment analysis software, I found that the resume had a lot of negative sentiments. analyzing the words in the resume, I noticed that there were a lot of words used in the document like risk, mitigation, etc. So, digging deeper into the resume, I noticed that this person had spent a good part of his career in the Risk Management Department of a Bank.**

Instead of creating a word cloud, we will create a comparison cloud. For creating a comparison cloud, we need the frequency counts on at least two variables. So, we will generate a word-wise sentiment score using the following code as shown:

```
library(reshape2)  
  
v_tidy_data %>%  
inner_join(get_sentiments("bing")) %>%  
count(word, sentiment, sort = TRUE) %>%  
acast(word ~ sentiment, value.var = "n", fill = 0)
```

```

## Joining, by = "word"
##                               negative positive
## abominable                  14      0

## abominably                 1      0
## abound                      0      7
## abrupt                      1      0
## abruptly                     1      0
## absence                      55      0
## absurd                       4      0
## abundance                     0      13
## abundant                     0      4
## abuse                        47      0
## abused                        7      0
## abuses                        11      0
## accessible                    0      2
## accidental                     3      0
## accomplish                     0      3

```

Notice the use of the acast() function from the reshape2 library. We can generate the comparison cloud as shown in the following code:

```

library(wordcloud)

v_tidy_data %>%
inner_join(get_sentiments("bing")) %>%
count(word, sentiment, sort = TRUE) %>%
acast(word ~ sentiment, value.var = "n", fill = 0) %>%
comparison.cloud(colors = c("red", "dark green"), max.words = 100)
## Joining, by = "word"

```

The output is as shown in the following screenshot:



*Figure 7.5: Comparison cloud*

At first glance, we can see the words that are majorly used in the text and words that express positive and negative sentiment.

## Conclusion

In this chapter, we discussed about sentiment analysis. We started our discussion with what is sentiment analysis and its applications. We discussed the different approaches to conduct sentiment analysis, that is, knowledge-based approach, statistical approach, and hybrid approach. We dived into conducting sentiment analysis using the knowledge-based approach. We discussed the different knowledge bases, known as lexicons, available in R for sentiment analysis. We discussed the different contexts in which the different lexicons could be used for sentiment analysis.

Lastly, we went through the process of conducting sentiment analysis on the collection of the books by Shakespeare. We went through the process of obtaining the text, extracting the words out of the text, and applying the **bing** lexicon on the text to extract sentiments. We developed methods for creating various visualizations from the sentiment analysis that we conducted on the text. We discussed how the visualizations could be interpreted. During the process, we also discussed various functions available in R language from various R libraries.

In the next chapter, we will explore emotion analysis.

### Points to remember

and nrc are 4 lexicons used for sentiment analysis

The unnest\_tokens() function from the tidytext library can be used for extracting words from a text.

The semi\_join() function in the dplyr library is used as a filtering join.

The inner\_join() function in the dplyr library is used as a mutating join.

The comparison.cloud() function from the wordcloud library can be used to create a comparison cloud.

### Multiple choice questions

The nrc lexicon was developed by:

Saif Mohammad and Peter Turney

Tim Loughran and Bill McDonald

Saif Mohammad and Bill McDonald

None of these

In R, which of these operators returns the quotient?

/

%

Mod

%/%

Which of these **lexicons** can be used for emotion analysis?

afinn

bing

nrc

None of these

The `unnest_tokens()` function from the `tidytext` library can be used for extracting:

words

bigrams

trigrams

All of these

The `semi_join()` function is available in which library?

joins

dplyr

tidytext

None of these

## Answers to MCQs

A

D

C

D

B

## Questions

Analyze a document using the **afinn**

### Key terms

Natural Language Processing is an art and science which helps us to extract information from text and use it in our computations and algorithms.

## CHAPTER 8

### Emotion Analysis

In the previous chapter, we studied how to conduct sentiment analysis. We discussed how to determine positive and negative sentiments in a text. In this chapter, we will extend the idea to extracting emotions from a text.

We discussed in [Chapter 7: Sentiment Analysis](#) that the *lexicons* in R can be used to determine sentiments and emotions in a text. In this chapter, we will use the **nrc** lexicon to determine emotions in a text. The **nrc** lexicon contains a list of words which can be used to determine eight emotions and two sentiments in a text. We will learn to make use of it in this chapter.

Our programming goal in this chapter is to determine the most prevalent emotion in a text. We will formulate a method by which any given text can be analyzed for this determination. We expect that such text should be available in a file. However, the text could be available in any other form like streams, etc. The mechanism for the analysis of such text will remain the same.

We will assume that the text is available in a **TXT** file. We already discussed how to read PDF files in [Chapter 6: Shiny Application](#). You could find out the ways to read other file types, not discussed in the book, on your own.

## Structure

In this chapter, we will discuss the following topics:

Conducting emotion analysis

- Reading and cleaning data
- Word analysis
- Sentiment flow in the Text
- Finding the most prevalent emotion

Further analysis of the document

- Bigram analysis
- Trigram analysis

## Objectives

After studying this unit, you should be able to:

Understand the method for conducting emotion analysis of a document

## Conducting emotion analysis

We will dive into the process of conducting emotion analysis for a given text. Using the process discussed in this chapter, we should be able to analyze any text. For the discussion, I will read the text from a TXT file. However, this step could be replaced with reading the text from any other type of file.

For this discussion, I will use the data obtained from my resume. *As it is my resume, I have full rights on this* Besides conducting emotion analysis on this data, we will also discuss a few other analyses that can be conducted on a resume.

We use the following process for an initial screening of resumes in our organization. It is reasonable to state that the way people write their resumes reflects a fair number of details about the person. Such an analysis can result in a considerable reduction of the work involved in the recruitment process and thus, save a reasonable amount of dollars for an organization.

## Reading and cleaning data

The first step to conduct the analysis for emotion detection is to read the data. I have saved my resume as a text file.

We can read a text file using the following code:

```
v_text <- tibble(text = gsub("[^[:alnum:]]//'", "",  
read_lines("./Resume.txt")))
```

The preceding code creates a data frame with a column named **text** and the column contains each line of the text in a separate row. The **gsub()** function is used to remove any special characters like etc. from the text.

We can see the contents of the data frame **v\_text** in the following code:

```
nrow(v_text)  
## [1] 360  
  
tail(v_text, 10)  
## # A tibble: 10 x 1  
##   text  
##     
## 1
```

```
## 1 "Formerly General Manager Siemens Information Systems  
Ltd. India "  
## 2 "Mobile 91 98990 09846 Email  
manojzmanzconsultancycom"  
## 3 ""  
## 4 "Mr Sudipto Sinha Roy CEO BEAS Consulting Pvt Ltd.  
India"  
## 5 "Formerly Senior Vice President NIIT Ltd. India Mobile 91  
96410 18920 Email s...  
## 6 ""  
  
## 7 "Resume of Partha Majumdar Page 2 of 5"  
## 8 ""  
## 9 "Resume of Partha Majumdar Page 1 of 5"  
## 10 ""
```

You can see that the file has 360 lines of text.

Any document will contain a number of blank lines. The next step is to eliminate these blank lines. We can achieve this by using the `drop_empty_row()` function from the library The code is shown as follows:

```
library(textclean)  
v_text <- drop_empty_row(v_text)
```

We can check the number of lines remaining in the data frame using the `nrow()` function on the data frame `v_text` as shown in the following code:

```
nrow(v_text)
## [1] 233
nchar(v_text)
## text
## 27851
```

We can notice that there are 233 lines of text and 27,851 characters remaining in the data frame

We can see the contents of the data frame v\_text in the following code:

```
tail(v_text, 10)
## # A tibble: 10 x 1
##   text
##
## 1 "Formerly CIO Etisalat Misr Egypt Mobile 971 50 634 0703
##   Email gghamishotmai...
## 2 "      Ms Panida Dunsiriphaiboon VP Campaign
##   Management Department dTAC Thaila...
## 3 "      Email Panidaddtaccoth"
## 4 "Mr Manoj Zaizada CEO ManZ Consultancy Pvt. Ltd.
##   India"
## 5 "Formerly General Manager Siemens Information Systems
##   Ltd. India "
## 6 "Mobile 98990 09846 Email manojzmanzconsultancycom "
## 7 "Mr Sudipto Sinha Roy CEO BEAS Consulting Pvt. Ltd.
##   India"
## 8 "Formerly Senior Vice President NIIT Ltd. India Mobile 91
##   96410 18920 Email s..."
```

```
## 9 "Resume of Partha Majumdar Page 2 of 5"  
## 10 "Resume of Partha Majumdar Page 1 of 5"
```

We can see that all the blank lines in the text have been removed from the data frame.

## Removing stop words

In computing, stop words are words that are filtered out before or after the natural language data (text) is processed. Stop words typically refer to the most common words in a language.

In R, there are a number of libraries that provide a list of stop words. For our analysis, we will use the list of stop words provided in the **tidytext** library. The stop words in the **tidytext** library are from three lexicons – onix, smart, and snowball. In the **tidytext** library, the stop words are available in the data frame

We can view the **stop\_words** data frame as follows:

```
stop_words
## # A tibble: 1,149 x 2
##   word      lexicon
##
## 1 a          SMART
## 2 a's         SMART
## 3 able        SMART
## 4 about       SMART
## 5 above       SMART
## 6 according   SMART
## 7 accordingly SMART
## 8 across      SMART
## 9 actually    SMART
```

```
## 10 after      SMART  
## # ... with 1,139 more rows
```

To remove the stop words from the text, we use the **anti\_join()** function from the **dplyr** library as shown in the following code:

```
v_tidy_data <- v_text %>%  
unnest_tokens(word, text) %>%  
anti_join(stop_words)  
## Joining, by = "word"  
head(v_tidy_data)  
## # A tibble: 6 x 1  
##   word  
## 1 partha  
## 2 majumdar  
## 3 flat  
## 4 sf  
## 5 o9  
## 6 sneha
```

We know from [Chapter 7: Sentiment Analysis](#) that we can extract all the words from a text using the **unnest\_tokens()** function from the **tidytext** library.

**When any join operation is performed, we must pass two data sets. We refer to the first data set as the left data set and the second data set as the right data set.**

**Anti-join** is a filtering join which returns all the rows from X with no matching values in Y, keeping just the columns from X. Here, X is the left data set and Y is the right data set.

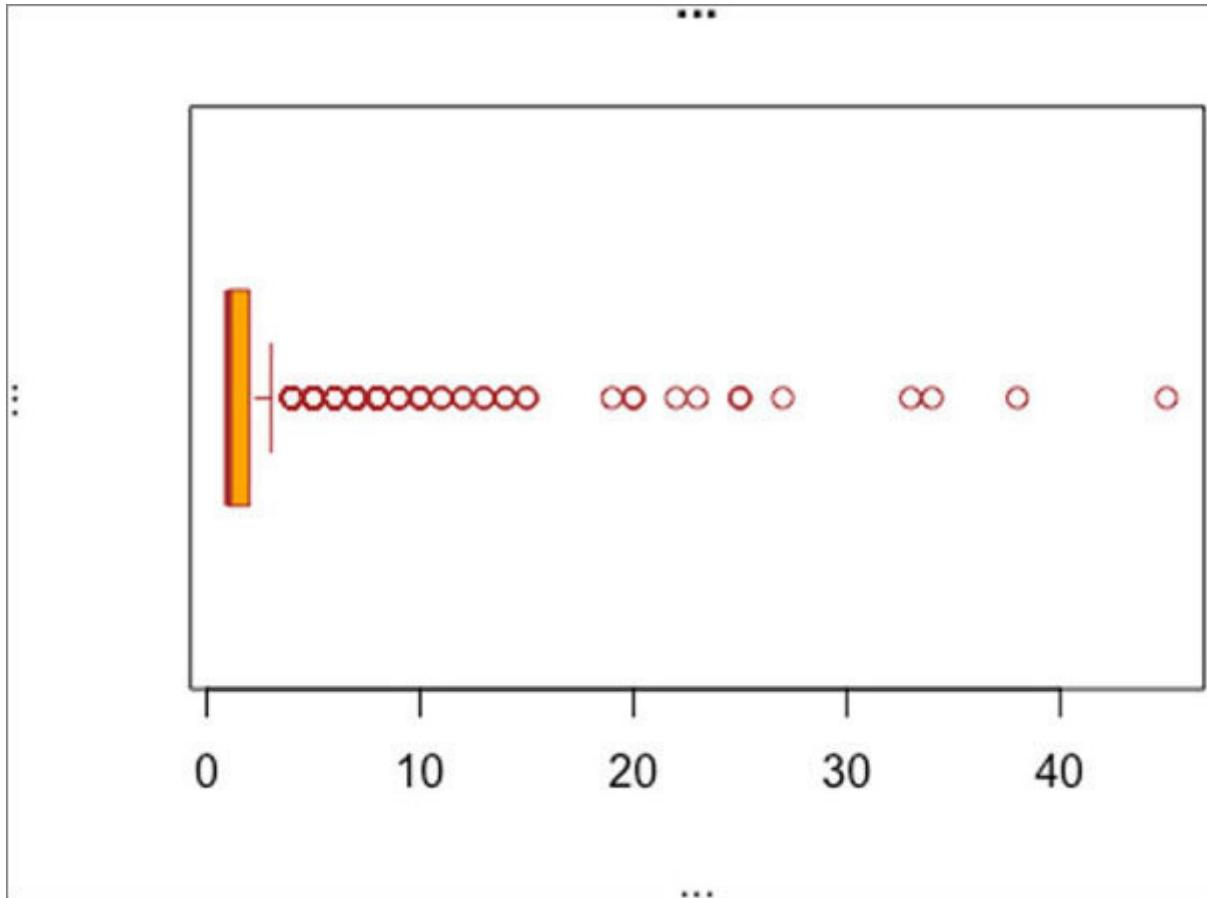
## Word analysis

Now that we have all the words from the text, we will conduct word analysis. We will generate some statistics from the words extracted.

### Check diversity of word frequencies

First, we will create a box plot to check the diversity of the word frequencies. The following is the code:

```
v_word_count <- v_tidy_data %>%
  count(word, sort = TRUE)
  boxplot(v_word_count$n,
  main = "...",
  xlab = "...",
  ylab = "...",
  col = "orange",
  border = "brown",
  horizontal = TRUE,
  notch = FALSE)
```



**Figure 8.1:** Box plot of word frequencies

We notice that the median value for word frequency is close to 1. We know that the word frequency is always a positive integer. And the minimum word frequency can be 1 for a word to exist in a text. However, there are words which have a frequency in excess of 30 as well. So, we must have a few words which appear several times in the text. We can confirm this by taking the following counts:

```
v_word_count <- v_tidy_data %>%
  count(word, sort = TRUE)
```

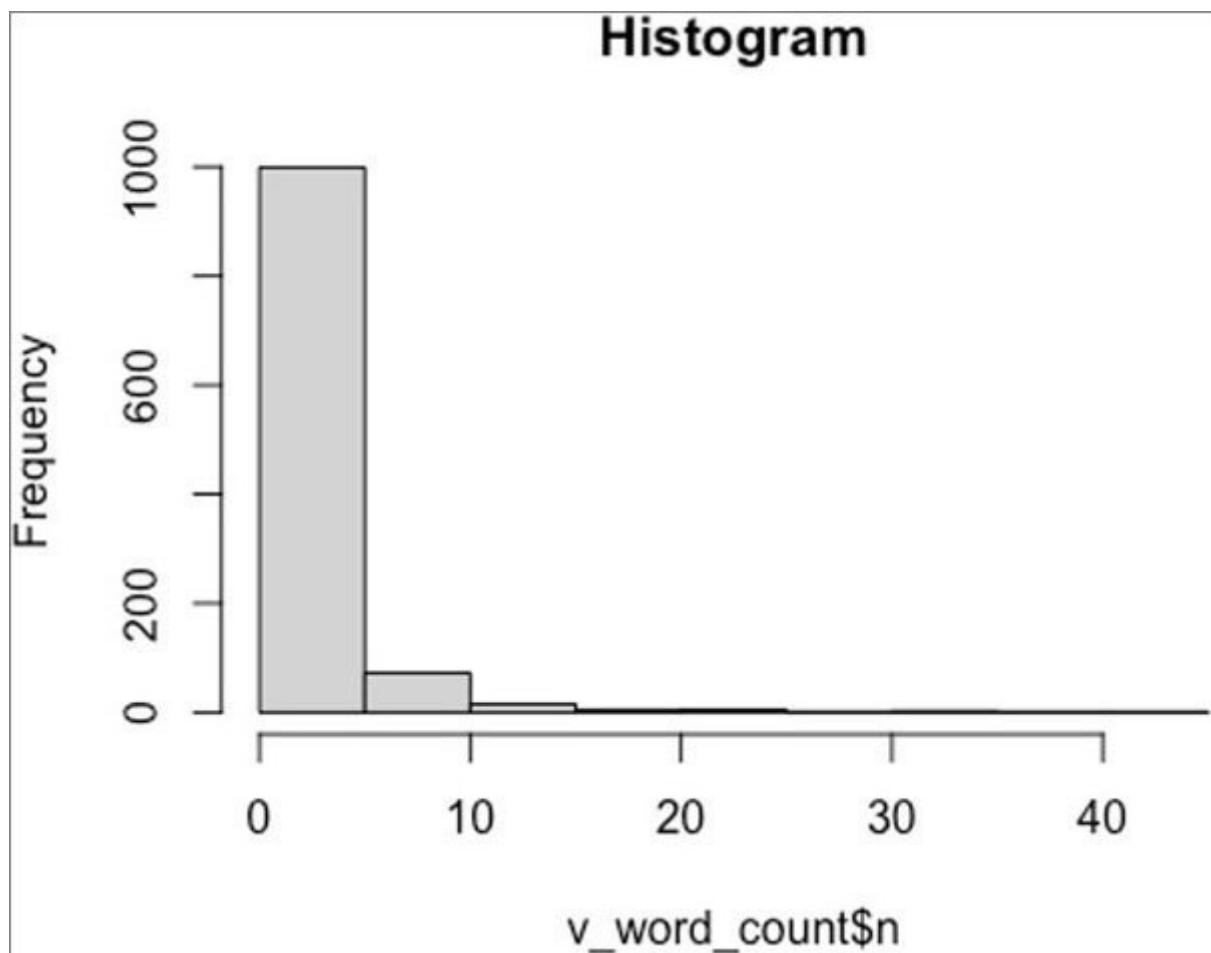
```
nrow(v_word_count)
## [1] 1100
```

```
nrow(v_word_count[which(v_word_count$n > 10),])  
## [1] 29
```

We notice that there are altogether 1,100 words in the text. And only 29 of these words have a frequency of more than 10. So, we should be wary that a few words can swing the most prevalent emotions extracted from this text. However, we cannot draw this conclusion at this stage as the same word may emote different emotions in different contexts.

We can visualize this data using a histogram shown as follows:

```
hist(v_word_count$n,  
main = "Histogram"  
)
```

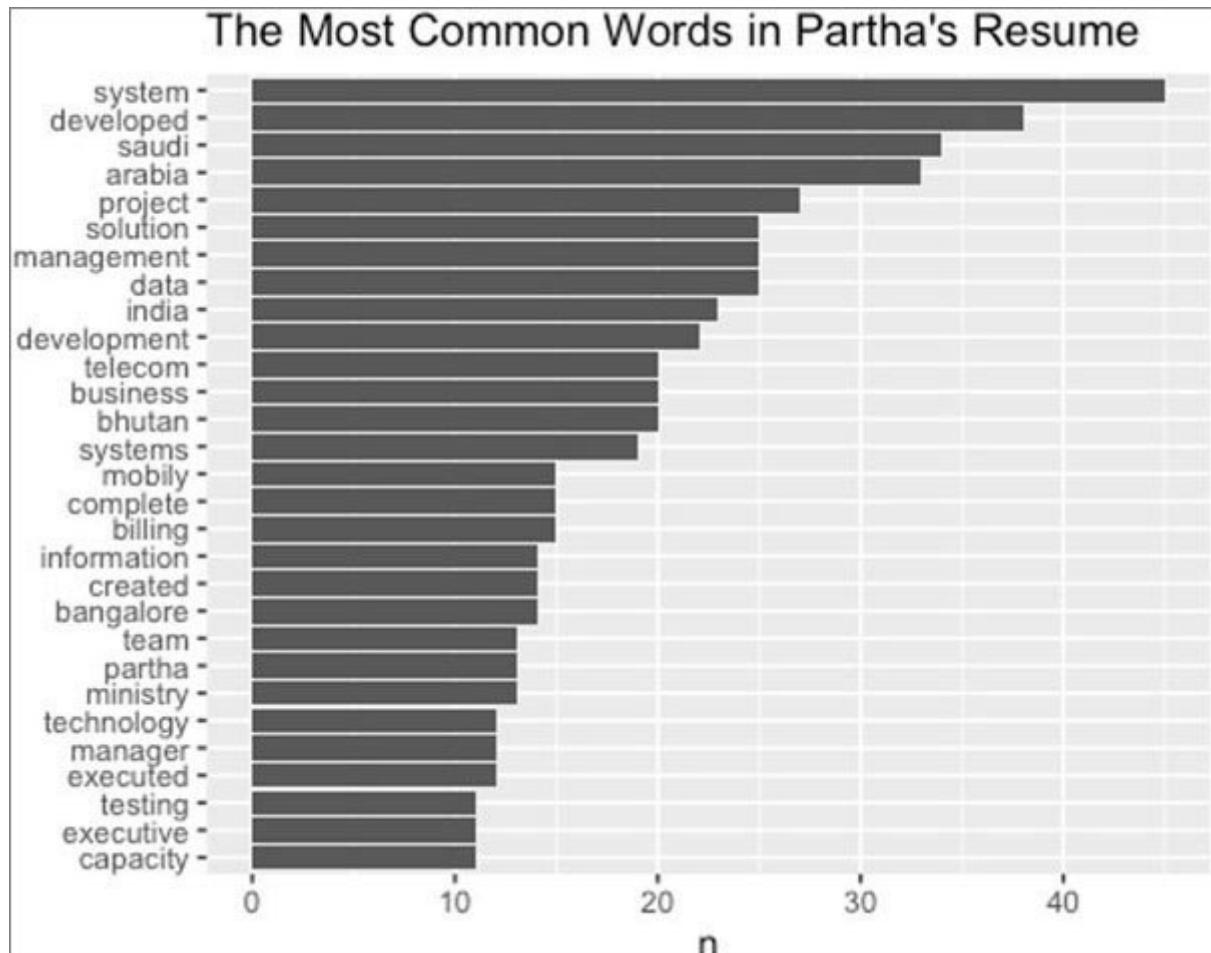


*Figure 8.2: Histogram of word frequencies*

## Most frequently used words in the text

Let us look at the most frequently used words in the text. We can generate this visualization using the following code:

```
v_tidy_data %>%
  count(word, sort = TRUE) %>%
  filter(n > 10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip() +
  ggtitle(paste("The Most Common Words in", v_document_title, sep
= " "))
```



**Figure 8.3:** Most frequent words in the text

It seems like this person has worked on several systems and been involved with the development of the systems. Also, the person seems to have spent a lot of time in Saudi Arabia, India, Bhutan, and Bengaluru. The person has been involved with telecom systems and worked reasonably with ministries. Also, we notice that the person has used his first name more than 10 times in the text. The people in the HR department can deduce many aspects from such an analysis.

A better visualization for the same analysis could be to generate the word cloud shown as follows:

```
wordcloud(words = v_word_count$word, freq = v_word_count$n,  
min.freq = 1,  
max.words = 250,  
random.order=FALSE,  
colors=brewer.pal(8, "Dark2"))
```



**Figure 8.4:** Word cloud of words in the text

Notice that we could have removed the numbers from the text.

## Emotions expressed by the words – visualization 1

Now that we have an idea of the words that have been used in the text, we will then analyze the sentiments and the emotions expressed by the words in the text.

As the first step, we can check for words that express positive sentiments. We have done this analysis in [Chapter 7: Sentiment Analysis](#) and the method are very similar. The difference is that in [Chapter 7: Sentiment](#) we used the bing lexicon and now we will use the nrc lexicon. In the bing lexicon, the words were classified in only two sentiments. However, in the nrc lexicon, we have eight emotions and two sentiments. So, we will create a grouping of positive sentiments and negative sentiments shown as follows:

```
v_positive_emotions <-  
c("positive","joy","anticipation","surprise","trust")  
v_negative_emotions <-  
c("negative","anger","disgust","fear","sadness")
```

Now, we can conduct a semi join to extract words with positive (or negative) sentiments shown as follows:

```
v_lexicon <- "nrc"  
v_positive_sentiment <- get_sentiments(v_lexicon) %>%  
filter(sentiment %in% v_positive_emotions)  
v_tidy_data %>%  
semi_join(v_positive_sentiment) %>%  
count(word, sort = TRUE)  
## Joining, by = "word"
```

```

## # A tibble: 132 x 2
##   word          n
## 
##   1 system      45
##   2 management  25
##   3 solution    25
##   4 information 14
##   5 ministry    13
##   6 team        13
##   7 technology   12
##   8 architecture 10
##   9 assurance    9
##  10 director     8
## # ... with 122 more rows

```

Notice that I used the variable **v\_lexicon** so that it can be changed easily across the program with only a single change. Also, notice that we used the operator **%in%** from the **dplyr** library. The **%in%** operator matches each value in the sentiment against all the values in the vector. It is an **OR** operation.

We see that, in the text, there are 132 words which espouse positive emotions. Also, we have their frequencies. The frequencies obtained are the frequencies of the words when they were used to express a positive emotion.

**The same word can be used to express a positive emotion in one context and a negative emotion in a different context.**

Similarly, we can generate the list of words which express negative emotions.

We can now generate the sentiment score of each word using an **inner join** with the lexicon shown as follows:

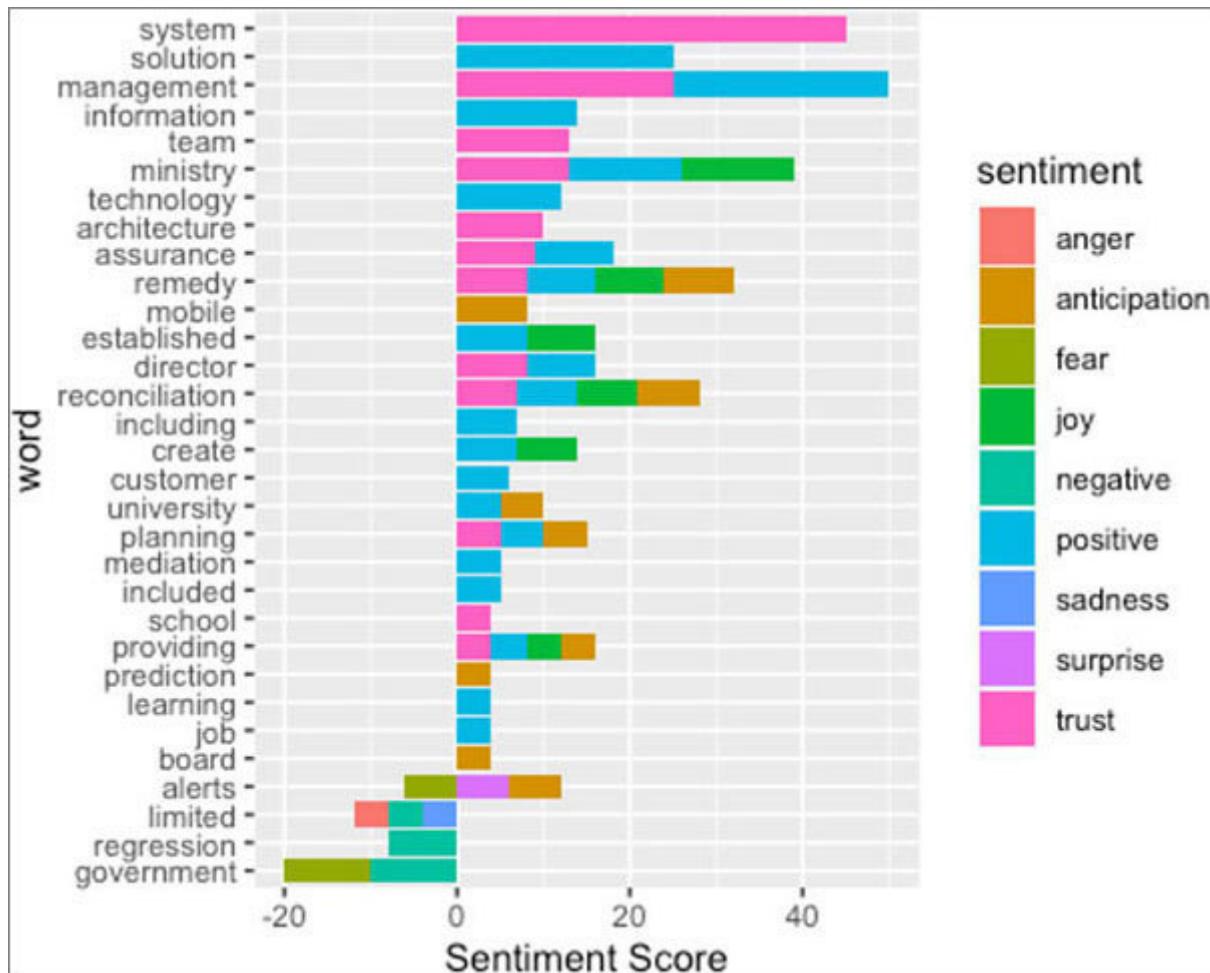
```
v_word_count <- v_tidy_data %>%
  inner_join(get_sentiments(v_lexicon)) %>%
  count(word, sentiment, sort = TRUE)
## Joining, by = "word"
head(v_word_count, 15)
## # A tibble: 15 x 3
##   word      sentiment     n
##   <chr>    <chr>       <dbl>
## 1 system    trust        45
## 2 management positive    25
## 3 management trust       25
## 4 solution   positive    25
## 5 information positive   14
## 6 ministry   joy         13
## 7 ministry   positive    13
## 8 ministry   trust        13
## 9 team       trust        13
## 10 technology positive   12
## 11 architecture trust      10
## 12 government fear        10
## 13 government negative    10
## 14 assurance  positive     9
## 15 assurance  trust        9
```

We can see the number of times that each word expresses an emotion in the text. See the example of the word management in the preceding output. It expressed positive sentiments in 25 contexts, and it expressed the emotion of trust in 25 contexts.

We can visualize this output using the **ggplot()** function from the **ggplot2** library shown as follows:

```
v_word_count %>%
filter(n > 3) %>%
mutate(n = ifelse(sentiment %in% v_negative_emotions, -n, n))
%>%
mutate(word = reorder(word, n)) %>%
ggplot(aes(word, n, fill = sentiment)) +
geom_col() +
coord_flip() +
labs(y = "Sentiment Score")
```

Notice that I have extracted words which have a frequency greater than 3. Also, notice that I have made the frequencies of words with a negative emotion as a negative number so that the positive emotions and the negative emotions appear in different directions of the axis:



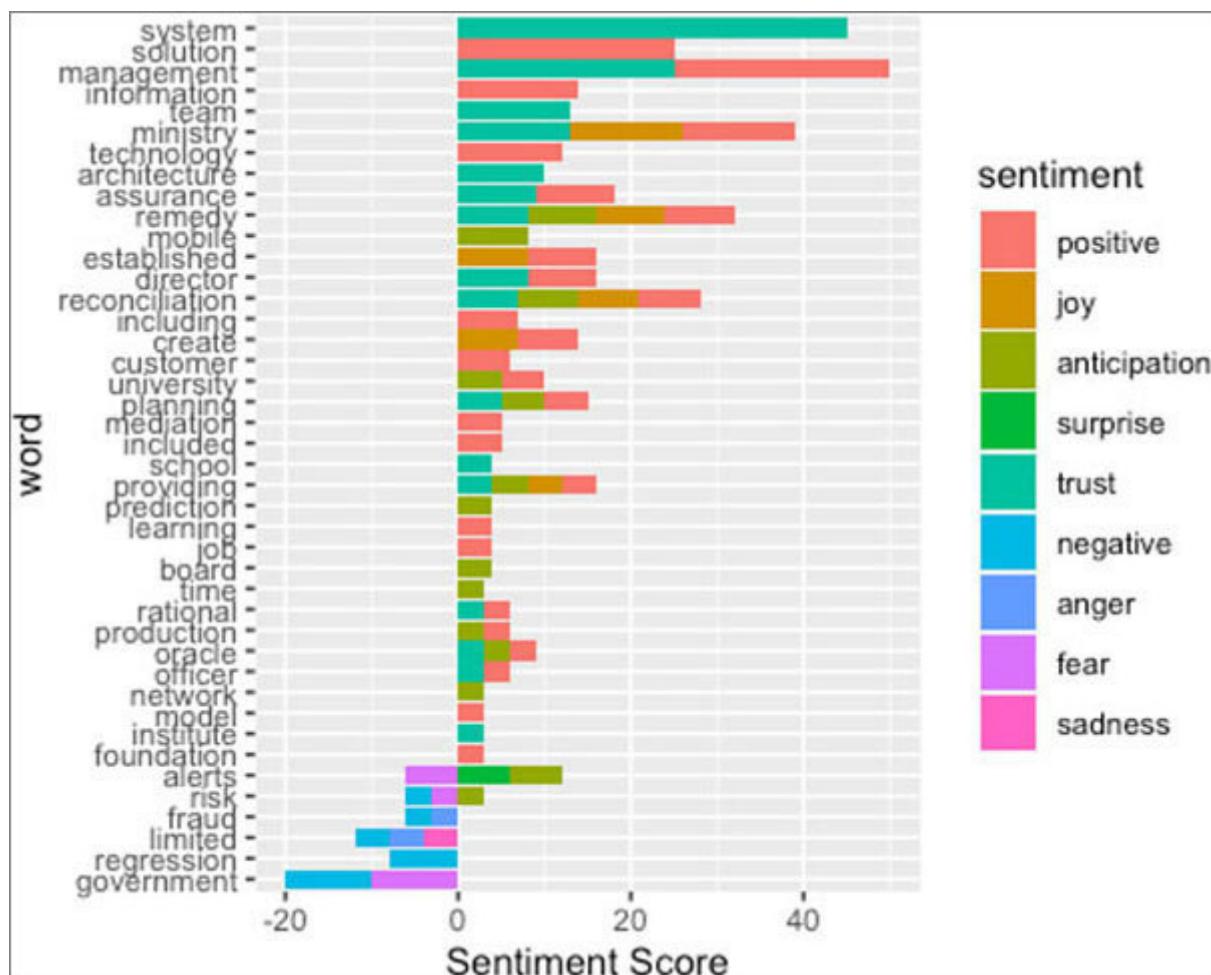
**Figure 8.5:** Sentiment scores of the words in the text

Notice the word alerts is used in both positive and negative contexts in the text.

Notice in [figure](#) the legends for the emotions are listed in the alphabetical order. However, it would be better if the legends for the emotions are listed in groups of positive sentiments and negative sentiments. We could achieve this using the following code:

```
v_positive_emotions <- c("positive","joy","anticipation","surprise",
"trust")
v_negative_emotions <- c("negative","anger","disgust","fear",
"sadness")
v_word_count$sentiment <- factor(v_word_count$sentiment, levels =
c(v_positive_emotions, v_negative_emotions))
v_word_count %>%
filter(n > 2) %>%
mutate(n = ifelse(sentiment %in% v_negative_emotions, -n, n)) %
%>%
mutate(word = reorder(word, n)) %>%
ggplot(aes(word, n, fill = sentiment))+
geom_col() +
coord_flip() +
labs(y = "Sentiment Score")
```

The output generated is shown as follows:



**Figure 8.6:** Sentiment scores with legends grouped by positive and negative sentiments

## Emotions expressed by the words – visualization 2

We can visualize the word contribution to the different emotions grouped by emotions using the sequence of the following code.

We will go through it step by step:

```
v_text_sentiment <- v_tidy_data %>%
  inner_join(get_sentiments(v_lexicon)) %>%
  count(word, index = row_number() %/%
    40, sentiment) %>%
  mutate(v_original_n = n) %>%
  mutate(v_positive_negative = ifelse(sentiment %in%
    v_positive_emotions, 'positive', 'negative')) %>%
  mutate(v_original_sentiment = sentiment) %>%
  spread(sentiment, n, fill = 0)
## Joining, by = "word"
```

First, we will create chunks of 40 lines of the text and store the chunk number in the column index. We will create a column called **v\_positive\_negative** to store whether the word espouses positive sentiment or negative sentiment based on whether the classified emotion is positive or negative. We will store the original sentiment detected and the word frequency for that sentiment in the columns **v\_original\_sentiment** and respectively. Lastly, we will use the **spread()** function from the **tidyR** library to create individual columns for the emotion score for each word (we have discussed the **spread()** function in [Chapter 7: Sentiment](#)

The contents of the data frame are shown as follows after executing this command:

```
v_text_sentiment  
## # A tibble: 521 x 14  
  
##      word   index v_original_n v_positive_nega... v_original_sent...  
anger anticipation  
##  
## 1 achi...     12          1 positive  
joy           0          0  
## 2 achi...     12          1 positive  
positive       0          0  
## 3 achi...     12          1 positive  
trust          0          0  
## 4 advi...     15          1 positive  
trust          0          0  
## 5 alarm      11          1 negative  
fear           0          0  
## 6 alarm      11          1 negative  
negative        0          0  
## 7 alarm      11          1 positive  
surprise        0          0  
## 8 aler...     12          4 negative  
fear           0          0  
## 9 aler...     12          4  
positive       anticipation 0          4  
## 10 aler...    12          4 positive  
surprise        0          0  
## # ... with 511 more rows, and 7 more variables: fear , joy ,  
## #   negative , positive , sadness , surprise , trust
```

Notice that in the preceding output, there is a column for nine emotions against each word. The column for the emotion disgust is missing. This is because no word in the text has been classified under the emotion disgust. The absence of a column for an emotion can cause problems in the application that we will develop for emotion analysis. To ensure that every emotion has a column in the data frame, we will augment following the code. The purpose of the following code is to ensure that every emotion has a column in the data frame:

```
v_text_sentiment <- v_text_sentiment %>%
  mutate(positive = ifelse("positive" %in%
    colnames(v_text_sentiment), positive, 0)) %>%
  mutate(joy = ifelse("joy" %in% colnames(v_text_sentiment), joy, 0))
%>%
  mutate(surprise = ifelse("surprise" %in%
    colnames(v_text_sentiment), surprise, 0)) %>%
  mutate(trust = ifelse("trust" %in% colnames(v_text_sentiment),
    trust, 0)) %>%
  mutate(anticipation = ifelse("anticipation" %in%
    colnames(v_text_sentiment), anticipation, 0)) %>%
  mutate(negative = ifelse("negative" %in%
    colnames(v_text_sentiment), negative, 0)) %>%
  mutate(anger = ifelse("anger" %in% colnames(v_text_sentiment),
    anger, 0)) %>%
  mutate(disgust = ifelse("disgust" %in% colnames(v_text_sentiment),
    disgust, 0)) %>%
  mutate(sadness = ifelse("sadness" %in%
    colnames(v_text_sentiment), sadness, 0)) %>%
```

```
mutate(fear = ifelse("fear" %in% colnames(v_text_sentiment), fear,  
0)) %>%
```

```
mutate(sentiment = (positive+joy+surprise+trust+anticipation) -  
(negative+anger+disgust+sadness+fear)) %>%  
ungroup()
```

All this code does is to create a column for each emotion and assign the emotion score for the word to this column if the column for that emotion exists in the data frame; else it creates the column with a ZERO emotion score for that word.

The contents of the data frame are as shown as follows after executing this command:

```
v_text_sentiment  
## # A tibble: 521 x 16  
##   word    index v_original_n v_positive_nega... v_original_sent...  
##   anger anticipation  
##  
##   1 achi...     12      1 positive          joy  
##   2 achi...     12      1 positive          o  
##   3 achi...     12      1 positive          trust  
##   4 advi...     15      1 positive          trust  
##   5 alarm      11      1 negative          o  
##   fear        o          o
```

```

## 6 alarm    11      1 negative      negative
o          o

## 7 alarm    11      1 positive     surprise
o          o
## 8 aler...   12      4 negative     fear
o          o
## 9 aler...   12      4 positive     anticipation
o          o
## 10 aler...  12      4 positive     surprise
o          o
## # ... with 511 more rows, and 9 more variables: fear , joy ,
## # negative , positive , sadness , surprise , trust ,
## # disgust , sentiment

```

Notice that all the emotions have a column in the data frame. Now, we can visualize this data using the `ggplot()` function from the `ggplot2` library shown as follows:

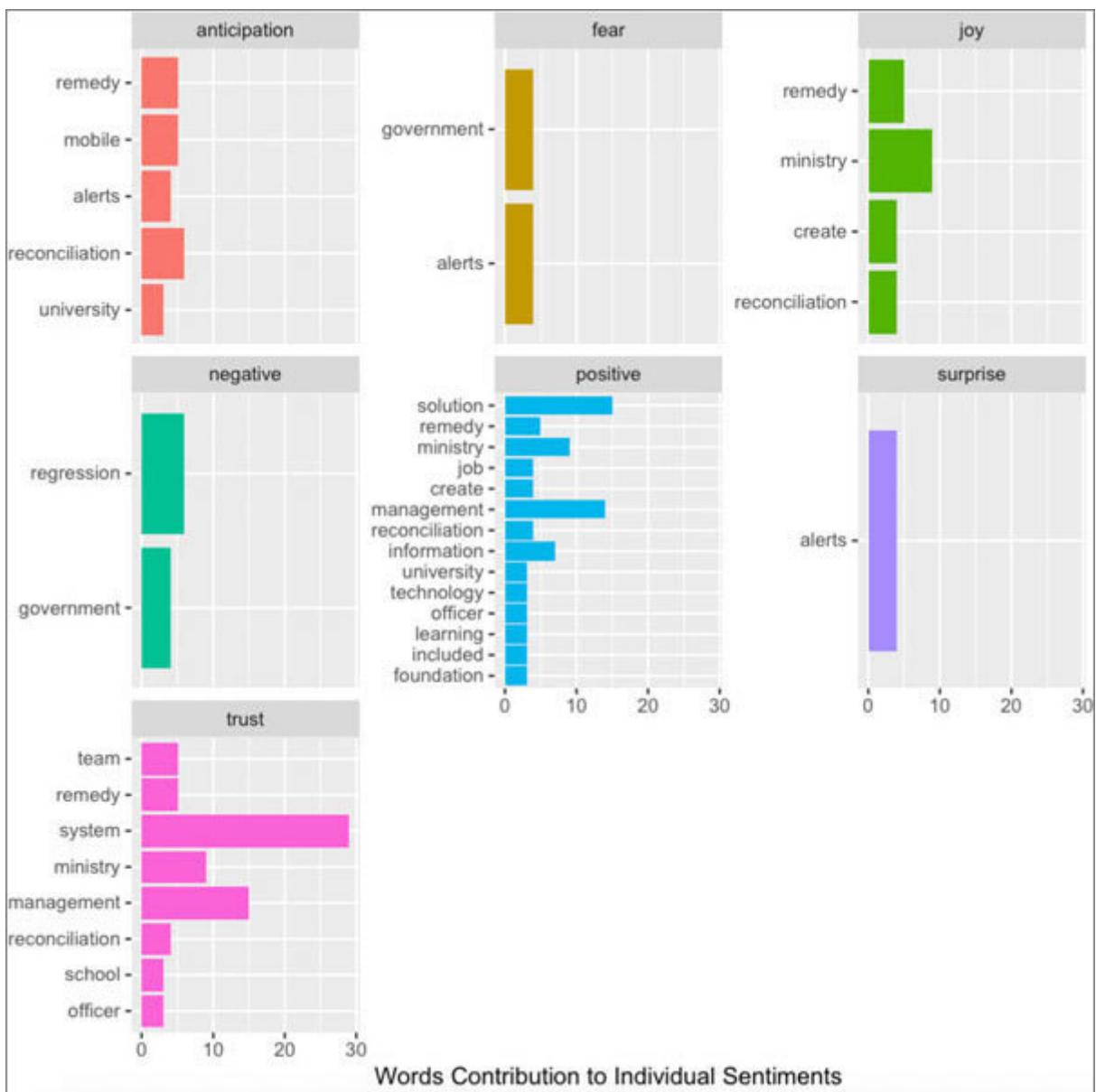
```

v_text_sentiment %>%
filter(v_original_n > 2) %>%
group_by(v_original_sentiment) %>%
top_n(5) %>%
ungroup() %>%
mutate(word = reorder(word, v_original_n)) %>%
ggplot(aes(word, v_original_n, fill = v_original_sentiment)) +
geom_col(show.legend = FALSE) +
facet_wrap(~v_original_sentiment, scales = "free_y") +
labs(y = "Words Contribution to Individual Sentiments", x = NULL) +
coord_flip()

```

## ## Selecting by sentiment

The following is the output:



**Figure 8.7:** Word contribution to emotions

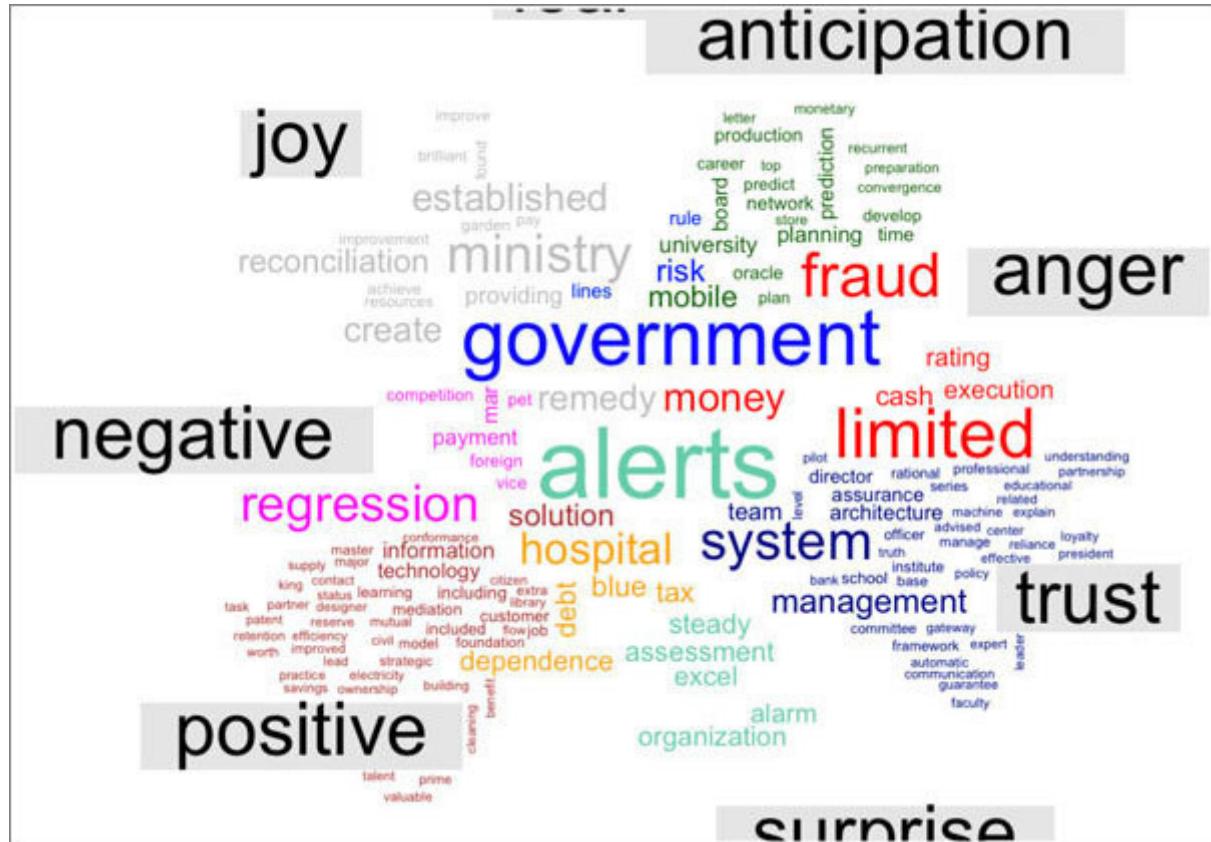
### Emotions expressed by the words – visualization 3

We can create another visualization of this data by creating a comparison cloud shown as follows. We have discussed this code in [Chapter 7: Sentiment](#)

```
v_tidy_data %>%
  inner_join(get_sentiments(v_lexicon)) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(
    colors = c("red", "dark green", "blue", "grey", "magenta", "brown",
    "orange", "mediumaquamarine", "navy"), max.words = 125)
```

Notice that, in the colors parameter, we must provide 10 colors for the 10 emotions, i.e., the 10 unique values in the column

The following is the output:



**Figure 8.8: Comparison cloud**

In making interpretations, we must be careful. For example, the word fraud is used in the resume in the context of having developed a system for fraud detection. However, the lexicon classifies this word as an expression of

## Sentiment flow in the text

Now, let us check the flow of the sentiments from the beginning to the end of the text. Remember that we have divided the text into chunks of 40 lines as otherwise, there will be too many lines to analyze and the analysis will not be meaningful. This is the same as what was done in [Chapter 7: Sentiment Analysis](#) where we had formed chunks of 80 lines.

We use the following code. This is like the parallel code that we used in [Chapter 7: Sentiment](#) except that we evaluate a vector in this case instead of a string:

```
v_text_nrc <- v_text_sentiment %>%
  mutate(v_pos_neg = ifelse(v_original_sentiment %in%
    v_positive_emotions, "positive", "negative")) %>%
  mutate(v_original_n = ifelse(v_pos_neg == "negative", -v_original_n,
    v_original_n))
```

The data frame **v\_text\_nrc** is shown as follows:

```
v_text_nrc
## # A tibble: 521 x 17
##   word index v_original_n v_positive_nega... v_original_sent...
##   <chr> <dbl>     <dbl>        <dbl>        <dbl>
## 1 anger      1         1           0           1
## 2 anticipation      2         1           1           0
## 3 ##
```

```

## 1 achi... 12 1 positive joy
o o
## 2 achi... 12 1 positive positive
o o
## 3 achi... 12 1
positive trust o o

## 4 advi... 15 1
positive trust o o
## 5 alarm 11 -1 negative fear
o o
## 6 alarm 11 -1 negative negative
o o
## 7 alarm 11 1 positive surprise
o o
## 8 aler... 12 -4 negative fear
o o
## 9 aler... 12 4 positive
anticipation o o
## 10 aler... 12 4 positive
surprise o o
## # ... with 511 more rows, and 10 more variables: fear , joy ,
## # negative , positive , sadness , surprise , trust ,
## # disgust , sentiment , v_pos_neg

```

We can visualize the result using the following code:

```

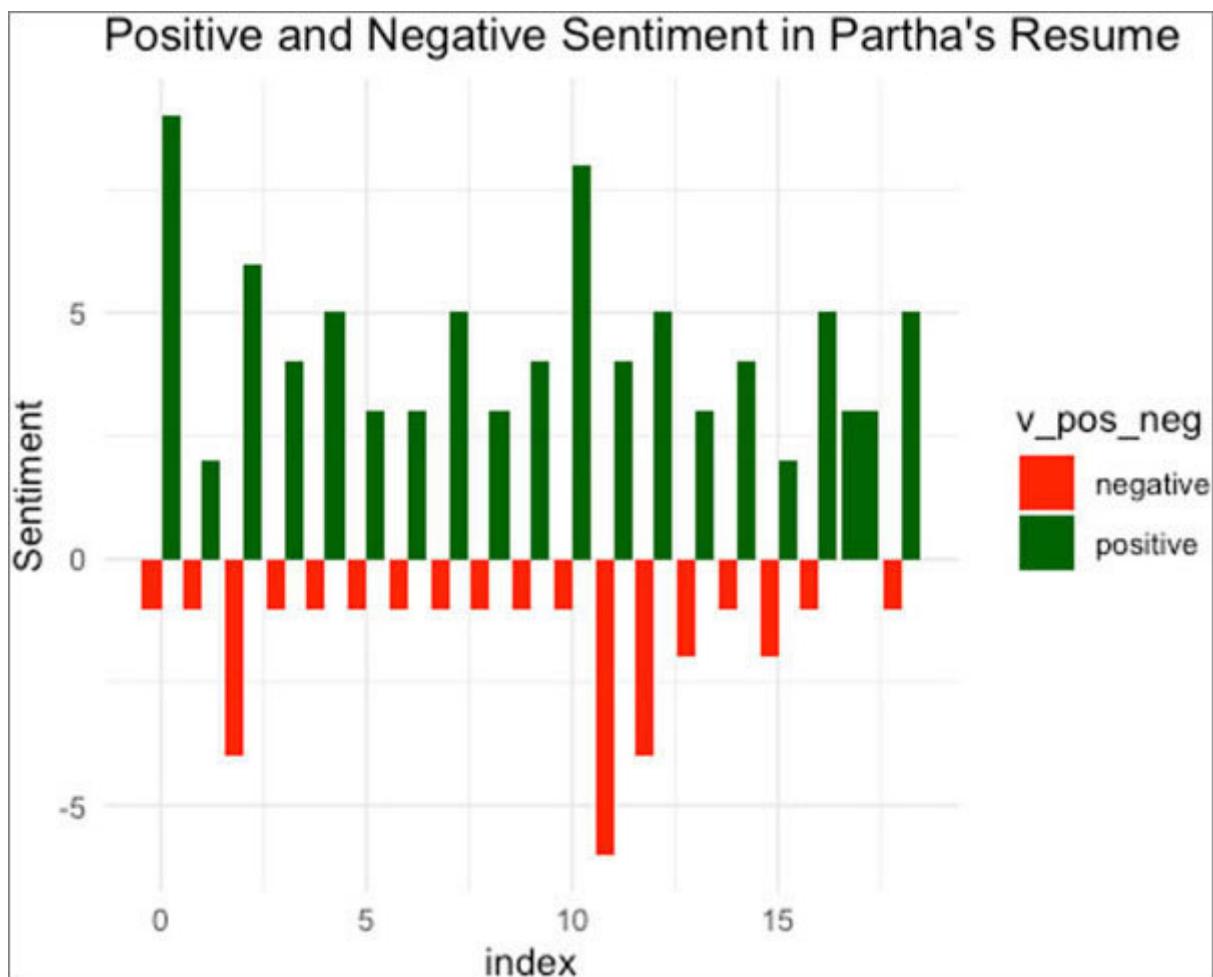
ggplot(data = v_text_nrc, aes(x = index, y = v_original_n, fill =
v_pos_neg)) + geom_bar(stat = 'identity', position =
position_dodge()) +
theme_minimal() +

```

```

ylab("Sentiment") +
ggtitle(paste("Positive and Negative Sentiments in",
v_document_title, sep = " ")) +
scale_color_manual(values = c("red", "dark green")) +
scale_fill_manual(values = c("red", "dark green"))

```



**Figure 8.9:** Positive and negative sentiments in the text from start to end

By analyzing this chart, the author can find contents with negative emotions and take corrective measures if required. This can be important for a document like a resume. For example, there seems to be something negative at the beginning of the

document and this may not auger well with HR managers given that this is an analysis of a resume.

## Finding the most prevalent emotion in the text

We have reached the climax of our analysis of a document. We will now determine the most prevalent emotion expressed in the document. This can be considered as the executive summary of emotion analysis of a document before digging deeper into individual analysis.

To determine the most prevalent emotion, our starting point is the **v\_text\_nrc** data frame that we created in the previous section. From this data frame, we will take only the columns which contain the scores of each emotion:

```
v_emotions <- v_text_nrc %>%
  select(index, anger, anticipation, disgust, fear, joy, sadness,
  surprise, trust)
head(v_emotions, 10)
## # A tibble: 10 x 9
##   index    anger anticipation disgust   fear     joy sadness
##   <dbl>     <dbl>        <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 1         12          0        0        0        0        1
## 2 2         12          0        0        0        0        0        1
## 3 3         12          0        0        0        0        0        1
## 4 4         12          0        0        0        0        0        1
## 5 5         12          0        0        0        0        0        1
## 6 6         12          0        0        0        0        0        1
## 7 7         12          0        0        0        0        0        1
## 8 8         12          0        0        0        0        0        1
## 9 9         12          0        0        0        0        0        1
## 10 10        12          0        0        0        0        0        1
```

```

## 4 15 o o o o 1
o o o
## 5 11 o o o o 1
o o o

## 6 11 o o o o 1
o o o
## 7 11 o o o o 1
o o o
## 8 12 o o o o 1
o o o
## 9 12 o o o o 1
o o o
## 10 12 o o o o 1
o o o

```

We can see that we have the scores for each emotion in a separate column. We had created this using the **spread()** function in the **tidyverse** library. Now, we will put all the emotion scores in one column. We can do this using the **melt()** function in the **reshape2** library shown as follows:

```

v_emotions <- v_text_nrc %>%
  select(index, anger, anticipation, disgust, fear, joy, sadness,
surprise, trust) %>%
  melt(id = "index")

```

We keep the column **index** from the original data frame and collapse all the other columns into a single column. The following is the output:

```

head(v_emotions, 10)
##      index variable value

## 1     12    anger     o
## 2     12    anger     o
## 3     12    anger     o

## 4     15    anger     o
## 5     11    anger     o
## 6     11    anger     o
## 7     11    anger     o
## 8     12    anger     o
## 9     12    anger     o
## 10    12    anger     o

```

Now, we will rename the column names to something convenient using the **rename()** function from the **dplyr** library shown as follows:

```

v_emotions <- v_text_nrc %>%
  select(index, anger, anticipation, disgust, fear, joy, sadness,
surprise, trust) %>%
  melt(id = "index") %>%
  rename(linenumber = index, sentiment = variable, value = value)

```

```

head(subset(v_emotions, value != o))
##      linenumber sentiment value

## 2085       12        joy     1
## 2086       12        joy     1
## 2087       12        joy     1
## 2088       15        joy     1
## 2089       11        joy     1

```

```
## 2090      11      joy      1
```

We will now group all the emotions into distinct groups using the **group\_by()** function from the **dplyr** library shown as follows:

```
v_emotions_group <- group_by(v_emotions, sentiment)
```

The output is shown as follows. We see that the column **sentiment** is now a factor:

```
v_emotions_group
## # A tibble: 4,168 x 3
## # Groups:   sentiment [8]
##   linenumber sentiment value
## 
##   1       12    anger     o
##   2       12    anger     o
##   3       12    anger     o
##   4       15    anger     o
##   5       11    anger     o
##   6       11    anger     o
##   7       11    anger     o
##   8       12    anger     o
##   9       12    anger     o
##  10      12    anger     o
## # ... with 4,158 more rows
```

Now, we will generate the score for each emotion using the **summarise()** function shown as follows:

```
v_by_emotions <- summarise(v_emotions_group,  
values=sum(value))
```

```
v_by_emotions  
## # A tibble: 8 x 2  
##   sentiment    values  
  
## 1 anger          o  
## 2 anticipation   o  
## 3 disgust        o  
## 4 fear           o  
## 5 joy            521  
## 6 sadness         o  
## 7 surprise        o  
## 8 trust          o
```

Before we proceed further, we need to convert the sentiment column which is a factor to a character:

```
v_temp <- v_by_emotions %>%  
  mutate_if(is.factor, as.character)
```

Notice that we have used the **mutate\_if()** function from the **dplyr** library which does the conversion of a column from a factor to a character only if the column is a factor. The preceding command will convert all the columns in the data frame which meet this condition.

The following is the output:

```
v_temp <- v_by_emotions %>%
  mutate_if(is.factor, as.character)
v_temp
## # A tibble: 8 x 2
##   sentiment     values
## 
## 1 anger          o
## 2 anticipation   o
## 3 disgust        o
## 4 fear           o
## 5 joy            521
## 6 sadness         o
## 7 surprise        o
## 8 trust          o
```

Now, we can extract the most prevalent emotion in the text using the following code:

```
ifelse(nrow(subset(v_temp, values == max(values))) == 1,
  unname(unlist(subset(v_temp, values == max(values)))),
  "No Prevalent Emotion")
## [1] "joy"
```

This should be straightforward to understand. However, let us dig deep into the statement for the **TRUE** condition. The **subset()** function should be clear and the output is shown as follows:

```
subset(v_temp, values == max(values))
## # A tibble: 1 x 2
##   sentiment values
##   <chr>     <dbl>
## 1 joy          521
```

We notice that the output is a data frame as expected. We use the **unlist()** function to convert this to a set of vectors:

```
unlist(subset(v_temp, values == max(values)))
## sentiment     values
##       "joy"      "521"
```

We can now remove the column names using the **unnname()** function:

```
unnname(unlist(subset(v_temp, values == max(values))))
## [1] "joy" "521"
```

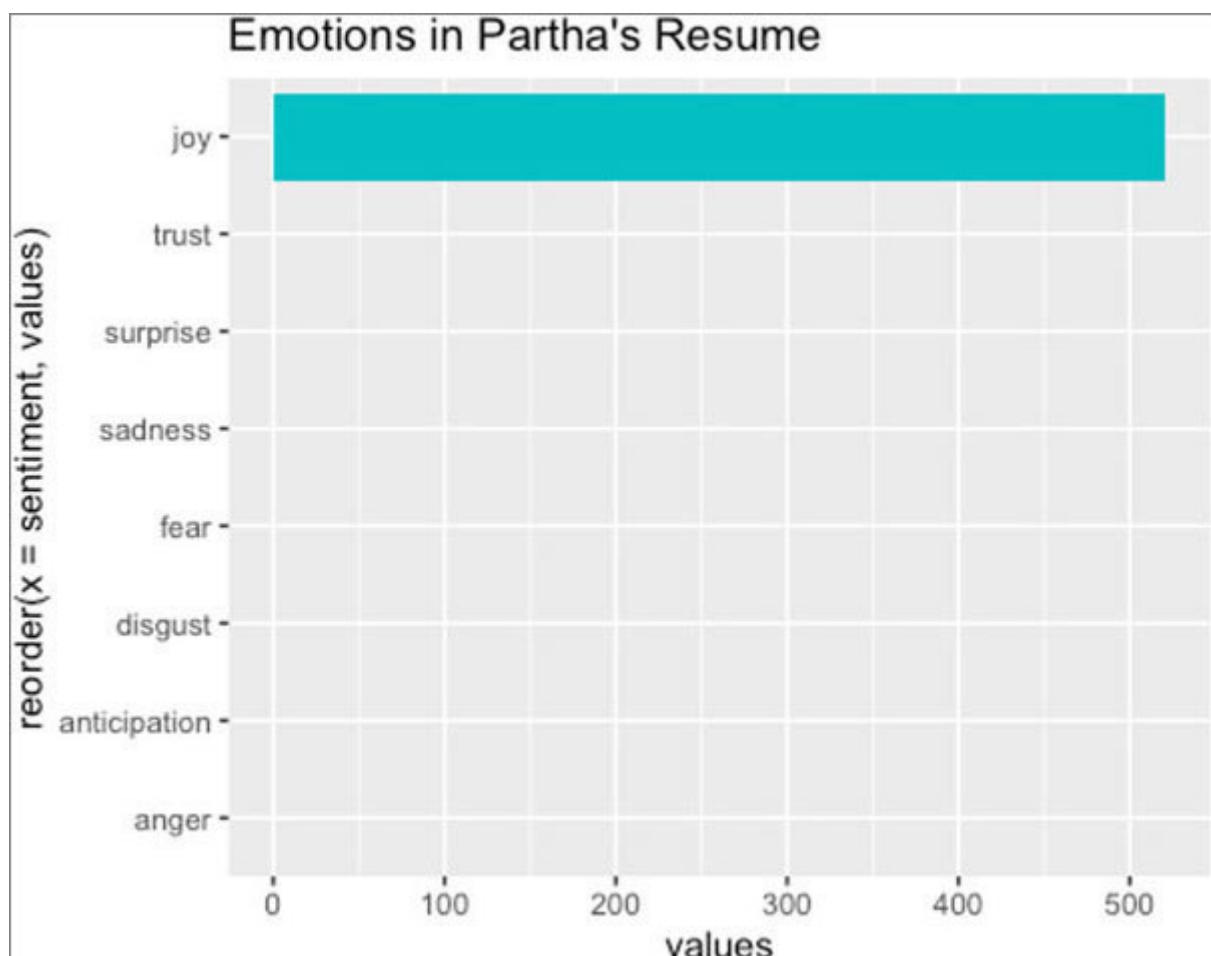
So, we obtain the most prevalent emotion in the text that we have already seen before:

```
ifelse(nrow(subset(v_temp, values == max(values))) == 1,
unnname(unlist(subset(v_temp, values == max(values)))),
"No Prevalent Emotion")
## [1] "joy"
```

We can visualize this using the **ggplot()** function from the **ggplot2** library as shown in the following code:

```
ggplot(aes(reorder(x=sentiment, values), y=values, fill=sentiment),  
       data = v_by_emotions) +  
  geom_bar(stat = 'identity') +  
  ggtitle(paste('Emotions in', v_document_title, sep = " ")) +  
  coord_flip() +  
  theme(legend.position = "none")
```

The following is the output:



**Figure 8.10:** Most prevalent emotion in the text

### Further analysis of the document

We discussed the core matter of this chapter. We will now discuss two more types of analyses that we can perform on the text. Especially, in the context of analyzing resumes, the analysis of bigrams and trigrams could be useful.

## Bigram analysis

Bigrams are the combinations of two consecutive words in a text. Bigrams are also referred to as 2-grams. The analysis of Bigrams can throw light on common word combinations used by the author. It can help us understand some characteristics of the author. This analysis is very useful in analyzing essays, project descriptions, etc.

We will go through the process of analyzing Bigrams. The first step is to extract the Bigrams from the text. We can achieve this using the **unnest\_tokens()** function from the **tidytext** library. The code is shown as follows:

```
v_text_bigrams <- v_text %>%  
  unnest_tokens(bigram, text, token = "ngrams", n = 2)  
v_text_bigrams  
## # A tibble: 3,982 x 1  
##   bigram  
##  
##   1 partha majumdar  
##   2 majumdar flat  
##   3 flat sf  
##   4 sf o9  
##   5 o9 sneha  
##   6 sneha sindhu  
##   7 sindhu apartments  
##   8 apartments no
```

```
## 9 no 25
## 10 25 amc
## # ... with 3,972 more rows
```

For the analysis, we need to remove the Bigrams formed in the combinations using stop words. So, we need to separate the words in the Bigrams and filter out the Bigrams containing stop words. To separate the words in the Bigrams, we can use the following code:

```
v_bigrams_separated <- v_text_bigrams %>%
separate(bigram, c("word1", "word2"), sep = " ")
```

The **separate()** function is available in the **tidyverse** library. The following is the output:

```
v_bigrams_separated
## # A tibble: 3,982 x 2
##   word1     word2
## 
## 1 partha    majumdar
## 2 majumdar  flat
## 3 flat       sf
## 4 sf         09
## 5 09        sneha
## 6 sneha     sindhu
## 7 sindhu    apartments
## 8 apartments no
## 9 no        25
## 10 25      amc
```

```
## # ... with 3,972 more rows
```

Now, we can apply the **filter()** function from the **dplyr** library to remove the Bigrams formed with the stop words shown as follows:

```
v_bigrams_filtered <- v_bigrams_separated %>%
  filter(!(word1 %in% stop_words$word)) %>%
  filter(!(word2 %in% stop_words$word))
v_bigrams_filtered
## # A tibble: 1,688 x 2
##   word1    word2
##   <chr>    <chr>
## 1 partha   majumdar
## 2 majumdar flat
## 3 flat     sf
## 4 sf       o9
## 5 o9       sneha
## 6 sneha   sindhu
## 7 sindhu  apartments
## 8 25       amc
## 9 amc      road
## 10 road    kb
## # ... with 1,678 more rows
```

Now, we can count the number of times each Bigram appears in the text shown as follows:

```
v_bigram_counts <- v_bigrams_filtered %>%
  count(word1, word2, sort = TRUE)
```

The following is the output:

```
v_bigram_counts
## # A tibble: 1,350 x 3
##   word1     word2       n
##   <chr>     <chr>     <dbl>
## 1 saudi     arabia      33
## 2 billing    system      7
## 3 mobily    saudi       7
## 4 data      reconciliation 6
## 5 hp         om          6
## 6 hutchinson australia   6
## 7 information systems    6
## 8 vodafone  hutchinson 6
## 9 bangalore  india       5
## 10 eduonix   simpliv     5
## # ... with 1,340 more rows
```

We can now visualize the Bigrams used in the text. We will use the **igraph** and **ggraph** libraries for this purpose using the following code:

```
library(igraph)
library(ggraph)

v_bigram_graph <- v_bigram_counts %>%
  filter(n > 2) %>%
  graph_from_data_frame()
```

```

v_bigram_graph
## IGRAPH a2a5obe DN-- 96 70 --
## + attr: name (v/c), n (e/n)
## + edges from a2a5obe (vertex names):
## [1] saudi      ->arabia      billing      ->system
## [3] mobily     ->saudi       data        -
>reconciliation

## [5] hp          ->om         hutchinson  -
>australia
## [7] information ->systems   vodafone   -
>hutchinson
## [9] bangalore   ->india      eduonix     ->simpliv
## [11] executive   ->manager    information  -
>technology
## [13] majumdar    ->consultancy management  ->system
## [15] reconciliation->activities remedy      ->itsm
## + ... omitted several edges

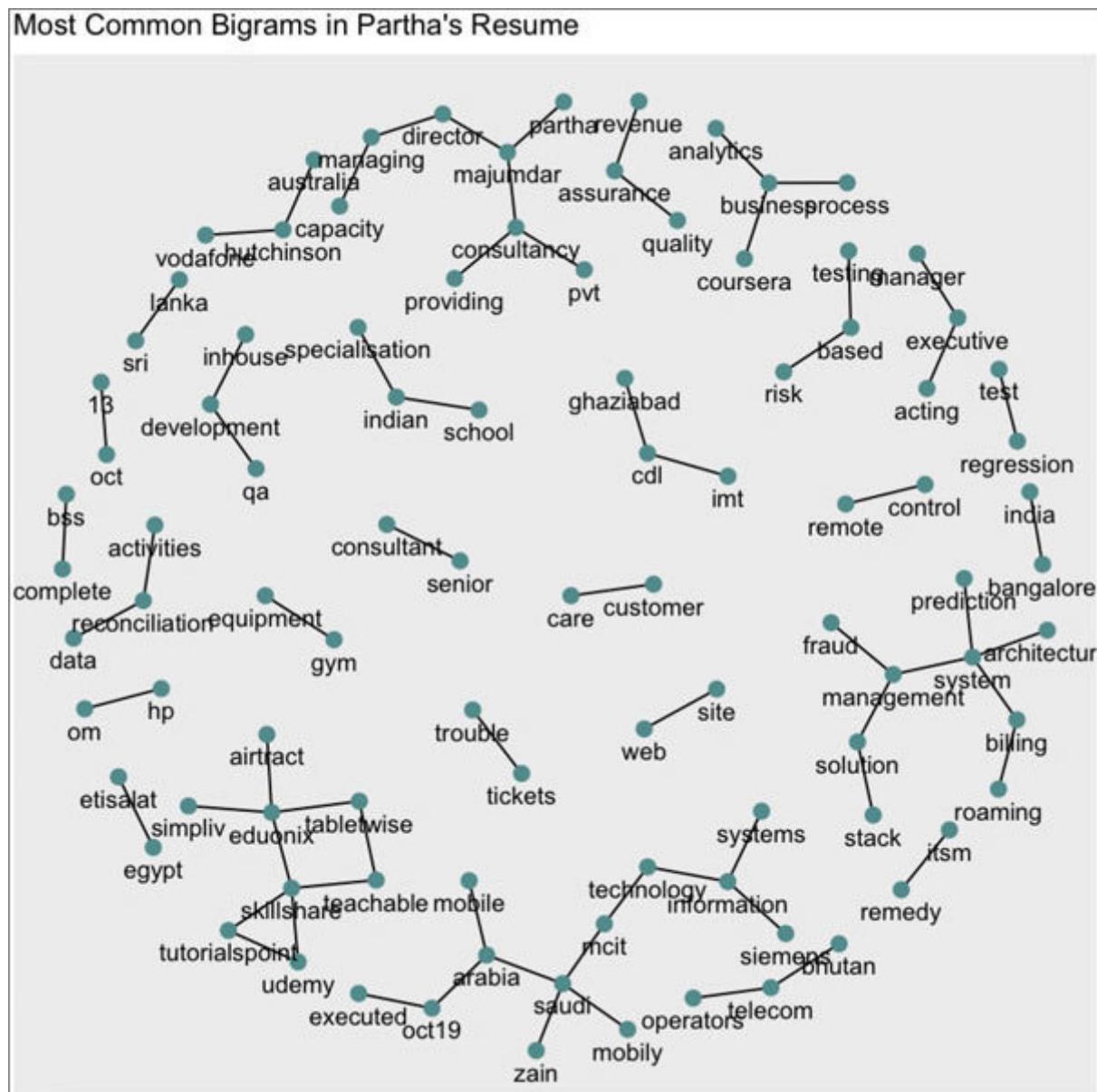
```

```

set.seed(2020)
ggraph(v_bigram_graph, layout = "kk") +
  geom_edge_link() +
  geom_node_point(color = "darkslategray4", size = 3) +
  geom_node_text(aes(label = name), vjust = 1.8) +
  ggtitle(paste("Most Common Bigrams in", v_document_title, sep =
" "))

```

The following is the output:



**Figure 8.11:** Bigrams in the text

We can visualize the frequency of the Bigrams. Before doing so, we need to reconstruct the Bigrams (as the words in the Bigrams have now been separated into two columns – **word1** and **word2**) and We will use the **unite()** function from the **tidyverse** library.

```
v_bigrams_united <- v_bigrams_filtered %>%  
  unite(bigram, word1, word2, sep = " ")
```

The following is the output:

```
v_bigrams_united
## # A tibble: 1,688 x 1

##     bigram
## 1 partha majumdar
## 2 majumdar flat
## 3 flat sf
## 4 sf o9
## 5 o9 sneha
## 6 sneha sindhu
## 7 sindhu apartments
## 8 25 amc
## 9 amc road
## 10 road kb
## # ... with 1,678 more rows
```

We will now count the number of occurrences of the Bigrams using the following code:

```
v_bigram_united_count <- v_bigrams_united %>%
  count(bigram) %>%
  filter(n > 3)
```

The **count()** function is enough to create the **frequency** table. However, we will add the **filter()** function to only extract the

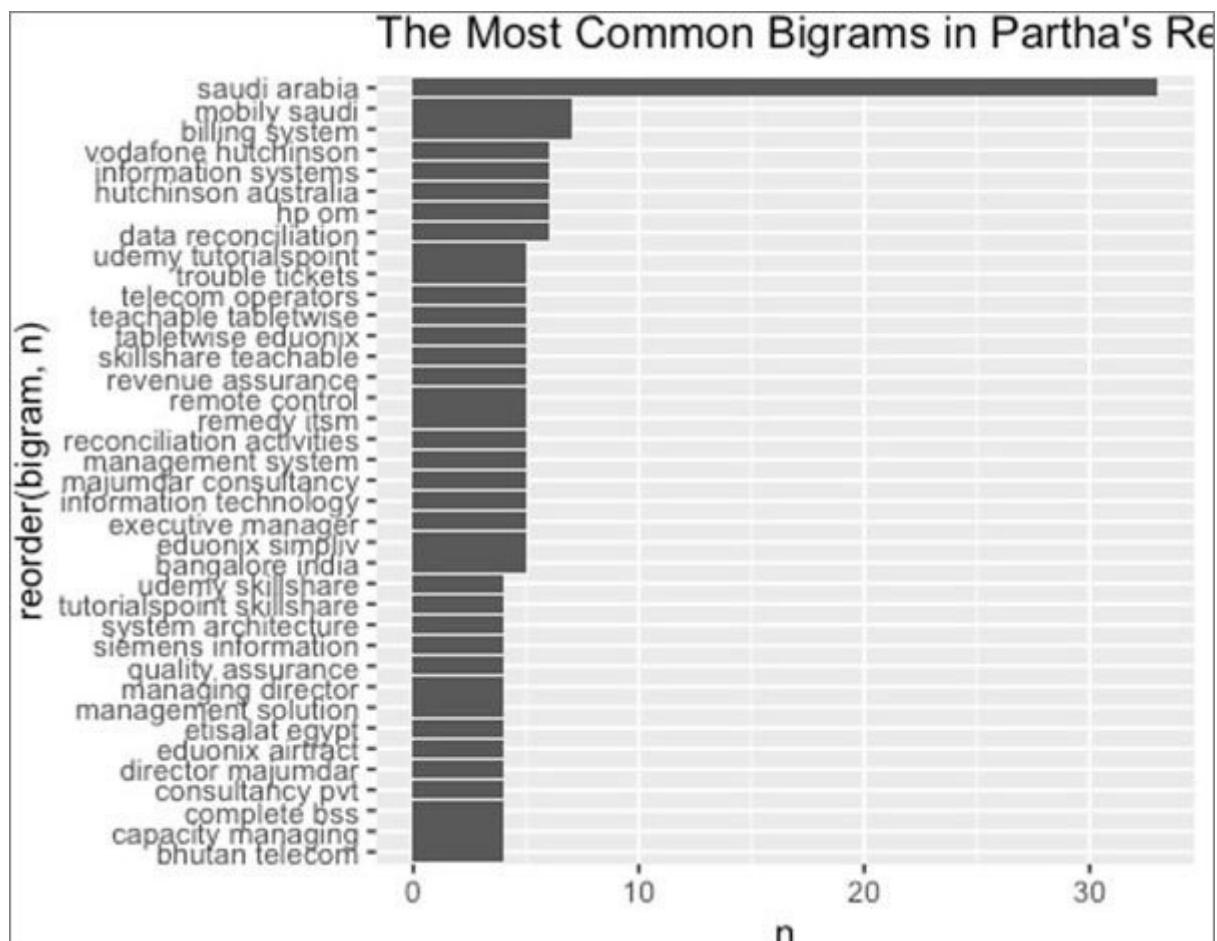
Bigrams occurring more than three times in the text. The output is shown as follows:

```
v_bigram_united_count
## # A tibble: 38 x 2
##   bigram          n
##   1 bangalore india      5
##   2 bhutan telecom       4
##   3 billing system      7
##   4 capacity managing   4
##   5 complete bss        4
##   6 consultancy pvt     4
##   7 data reconciliation 6
##   8 director majumdar   4
##   9 eduonix airtract    4
##  10 eduonix simpliv     5
## # ... with 28 more rows
```

Notice that we have not sorted the **frequency** table. We can visualize this **frequency** table using the **ggplot()** function shown as follows:

```
ggplot(aes(x = reorder(bigram, n), y=n), data =
v_bigram_united_count) +
geom_bar(stat = 'identity') +
ggttitle(paste("The Most Common Bigrams in", v_document_title,
sep = " ")) +
coord_flip()
```

The following is the output:



**Figure 8.12:** Most common bigrams in the text

We can create a word cloud of this frequency table for another visualization shown as follows:

```
wordcloud(words = v_bigram_united_count$bigram,  
freq = v_bigram_united_count$n,  
min.freq = 3,  
max.words = 250,  
random.order=FALSE,  
colors=brewer.pal(8, "Dark2"))
```

The following is the output:



**Figure 8.13:** Word Cloud of Bigrams in the text

## Trigram analysis

Trigrams are combinations of three consecutive words in a text. Trigrams are also referred to as 3-grams. Similar to Bigrams, the analysis of Trigrams can throw light on common word combinations used by the author.

We will go through the process of analyzing Trigrams. The first step is to extract the Trigrams from the text. We can achieve this using the **unnest\_tokens()** function from the **tidytext** library. The code is shown as follows:

```
v_text_trigrams <- v_text %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3)
```

```
head(v_text_trigrams, 10)
## # A tibble: 10 x 1
##   trigram
##   1 partha majumdar flat
##   2 majumdar flat sf
##   3 flat sf o9
##   4 sf o9 sneha
##   5 o9 sneha sindhu
##   6 sneha sindhu apartments
##   7 sindhu apartments no
##   8 apartments no 25
```

```
## 9 no 25 amc  
## 10 25 amc road
```

Like Bigrams, we need to remove the Trigrams formed using stop words. So, we need to separate the words in the Trigrams. We can do this using the **separate()** function shown as follows:

```
v_trigrams_separated <- v_text_trigrams %>%  
separate(trigram, c("word1", "word2", "word3"), sep = " ")
```

```
head(v_trigrams_separated, 10)  
## # A tibble: 10 x 3  
##   word1     word2     word3  
##     
## 1 partha    majumdar  flat  
## 2 majumdar  flat      sf  
## 3 flat       sf        09  
## 4 sf         09        sneha  
## 5 09        sneha     sindhu  
## 6 sneha     sindhu    apartments  
## 7 sindhu    apartments no  
## 8 apartments no        25  
## 9 no        25        amc  
## 10 25       amc      road
```

Now, we can filter out the Trigrams formed using stop words shown as follows:

```
v_trigrams_filtered <- v_trigrams_separated %>%
```

```

filter(!word1 %in% stop_words$word) %>%
filter(!word2 %in% stop_words$word) %>%
filter(!word3 %in% stop_words$word)

head(v_trigrams_filtered, 10)
## # A tibble: 10 x 3
##   word1    word2    word3
##
## 1 partha  majumdar flat
## 2 majumdar flat      sf
## 3 flat      sf      o9
## 4 sf        o9      sneha
## 5 o9        sneha  sindhu
## 6 sneha    sindhu apartments
## 7 25       amc     road
## 8 amc      road     kb
## 9 road     kb      sandra
## 10 kb      sandra   rt

```

Having filtered out the unwanted Trigrams, we can now combine the words forming the Trigrams using the **unite()** function shown as follows:

```

v_trigrams_united <- v_trigrams_filtered %>%
  unite(trigram, word1, word2, word3, sep = " ")

```

```

head(v_trigrams_united, 10)
## # A tibble: 10 x 1
##   trigram
##
## 1 parthamajumdarflat
## 2 majumdarflat      sf
## 3 flat      sf      o9
## 4 sf        o9      sneha
## 5 o9        sneha  sindhu
## 6 sneha    sindhu apartments
## 7 25       amc     road
## 8 amc      road     kb
## 9 road     kb      sandra
## 10 kb      sandra   rt

```

```

## 1 partha majumdar flat
## 2 majumdar flat sf
## 3 flat sf 09
## 4 sf 09 sneha
## 5 09 sneha sindhu
## 6 sneha sindhu apartments
## 7 25 amc road
## 8 amc road kb
## 9 road kb sandra
## 10 kb sandra rt

```

Now, we can generate the **frequency** table of the Trigrams shown as follows:

```

v_trigrams_united_count <- v_trigrams_united %>%
  count(trigram) %>%
  filter(n > 2)

```

trigram	n
1 acting executive manager	3
2 arabia oct19 executed	3
3 capacity managing director	4
4 data reconciliation activities	5
5 director majumdar consultancy	4
6 imt cdl ghaziabad	3
7 information technology mcit	3
8 majumdar consultancy pvt	4

```

## 9 managing director majumdar          4
## 10 mcit saudi arabia                 3
## # ... with 16 more rows

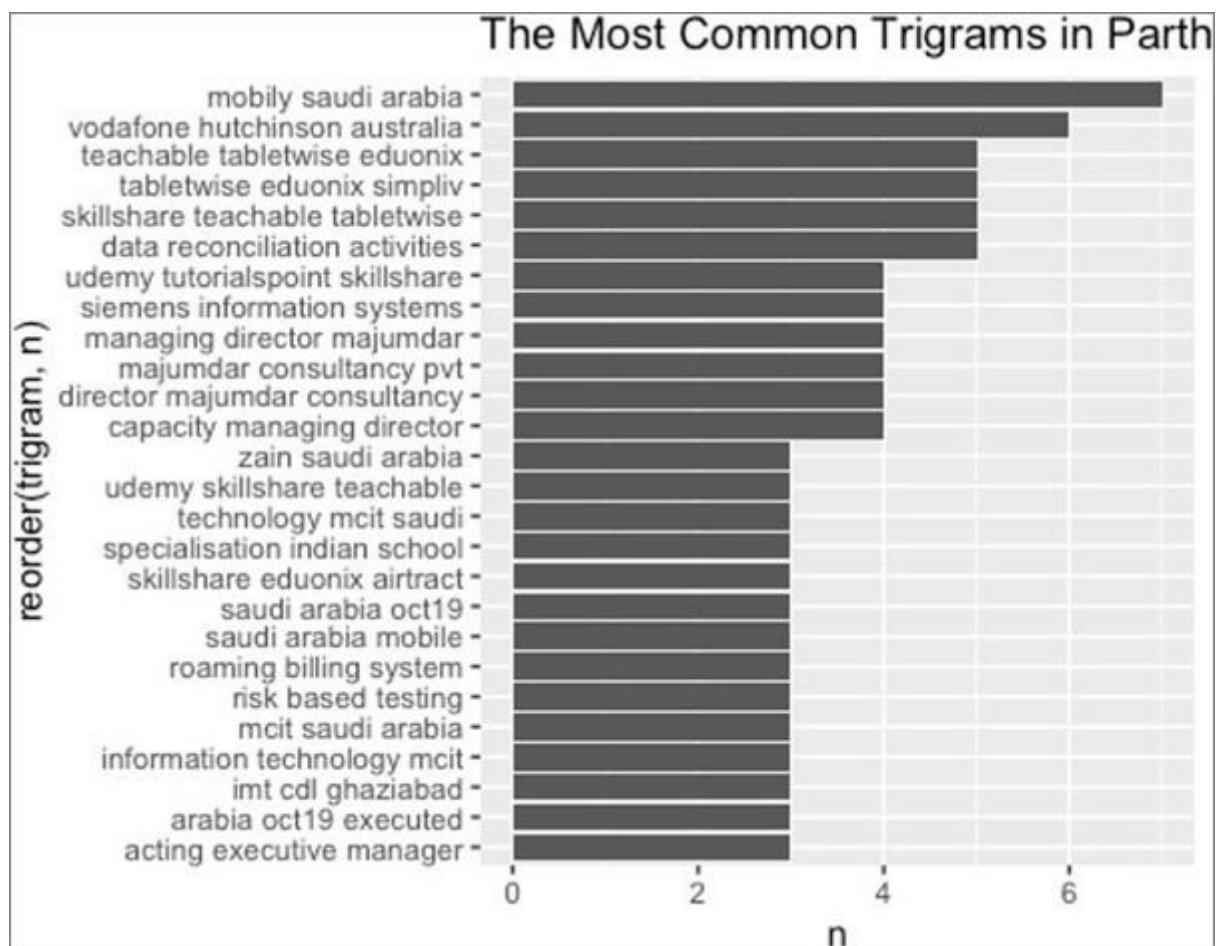
```

We can now visualize this **frequency** table of Trigrams shown as follows:

```

ggplot(aes(x = reorder(trigram, n), y=n), data =
v_trigrams_united_count) + geom_bar(stat = 'identity') +
ggtitle(paste("The Most Common Trigrams in", v_document_title,
sep = " ")) + coord_flip()

```



**Figure 8.14:** Most common Trigrams in the text



## Conclusion

In this chapter, we went through the process of conducting Emotion analysis for a text. We read the text from a TEXT File. However, the same process can be adopted for reading text from any other type of file.

Our sample text was a resume. We went through various analyses to ultimately extract the Most Prevalent Emotion expressed in the text. During the journey, we created several visualizations to make useful inferences from the analysis. We discussed a few types of inferences that interested people can make from such an analysis.

We concluded the chapter by conducting an analysis of Bigrams and Trigrams. In the next chapter, we will create a data product using the code that we discussed in this chapter.

### Points to remember

The **drop\_empty\_row()** function from the library **textclean** can be used to remove the empty rows.

Stop words are filtered out before or after the natural language data (text) is processed.

The nrc lexicon can be used to extract eight emotions and two sentiments.

Bigrams are combinations of two consecutive words from the text. They are also referred to as 2-grams.

Trigrams are combinations of three consecutive words from the text. They are also referred to as 3-grams.

### Multiple choice questions

The function to remove the column names is:

unlist

uname

Both of these

None of these

The **melt()** function is available in which library?

reshape2

tidytext

tidytext

None of these

The **spread()** function is available in which library?

reshape2

tidyverse

tidytext

None of these

The **separate()** and **unite()** functions are available in which library?

reshape2

tidyverse

tidytext

None of these

The **graph\_from\_data\_frame()** function is available in which library?

ggplot2

ggplot

igraph

ggraph

## Answers to MCQs

B

A

B

B

C

## Questions

What are the parameters to the **gsub()** function to only accept alphabets and spaces from a text?

Conduct an analysis of a financial document, like the annual report of a company, using the loughran lexicon.

## Key terms

n-grams are basically a set of co-occurring words within a given window. n-grams of texts are extensively used in text mining and natural language processing tasks.

In [Chapter 8: Emotion Analysis](#), we discussed the process for conducting emotion analysis. We went through various analyses to arrive at the most prevalent emotion expressed in a text. We saw various visualizations of a given text from different perspectives.

In this chapter, we will convert this analysis method into a Shiny Application. As Shiny applications can be accessed by anyone once published on the **World Wide Web** the tool we develop could be used by everybody to analyze any document for extracting the emotion expressed in it.

This will be the first large Shiny application we will discuss in this book. We will make use of all the concepts of Shiny application development discussed in [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#)

The complete code for the Shiny application is shared in this chapter. This code can be modified and beautified and enhanced for making this application much more attractive and useful.

## Structure

In this chapter, we will discuss the following topics:

### Design of ZEUSg

- Landing page
- Outputs from ZEUSg

### Programming ZEUSg

- Giving credits for Using a library
- Libraries Used for Developing ZEUSg
- Creating tabs within a tab
- Reading the input file

### The complete code of ZEUSg

- app.R
- ZEUSg.Rmd



## Objectives

After studying this unit, you should be able to:

Create Shiny applications for emotion analysis

## Design of ZEUSg

We will first discuss what we will develop and then go about developing the same. We will make use of all the components that we discussed in [Chapter 8: Emotion](#). So, when we discuss the code, it will only be about discussing the Shiny programming aspects.

### **WHY ZEUSG?**

I wanted a name for my application. As the application was about making predictions, I chose the name Zeus – the ruler of all Gods on Mount Olympus. However, when I added the letter number, I got 71 (= 26 + 5 + 21 + 19). Now, I prefer numbers which are multiples of 3. So, I added the letter ‘g’ in the end to get a score of 78. This was a multiple of 3 and (7 + 8) is 15 and (1 + 5) is 6, which is my lucky number.

ZEUSg is not a well-structured program. It does what it is intended to perform. However, it is also important for a program to be beautiful. I have done this intentionally as I use the code of ZEUSg to explain some Shiny Programming concepts. This is an exercise for you to put ZEUSg into a proper structure. You will come across a much better program in [Chapter 12:](#)

## Landing page

ZEUSg can be invoked by typing the URL <https://partha.shinyapps.io/EmotionAnalysis> in any browser. On invoking ZEUSg, we see the following page:



**Figure 9.1:** ZEUSg landing page

Throughout the book, for all the applications discussed, I have maintained the same layout that we first discussed in [Chapter 5: Shiny Application](#). This is so that we do not spend time discussing the layouts, which can take a lot of words. However, you can be creative in designing beautiful and complex UI/UX. Shiny applications can create most of what is possible to design.

So, we have a header panel on the top. Following the header panel are 2 panels, which I refer to as the left-hand panel and the right-hand panel. In the left-hand panel, we take the user inputs. In the right-hand panel, we display all the outputs based on the user inputs.

In the left-hand panel, we take a TEXT file as input from the user. We have a text box for taking an input for the document title. This document title is used in the report of the analysis created on this input file. Also, we have a **Download** button to download the report of the analysis created on the input file. You would have guessed that the report will be created using RMarkdown.

## Outputs from ZEUSg

Now, let us look at the outputs of ZEUSg. All the outputs are presented in the right-hand panel. There are two tabs in the right-hand panel of ZEUSg. The first tab contains a static image as can be seen in the landing page. Every time the user provides a new file for analysis, ZEUSg will revert to this tab. The second tab titled **ANALYSIS** contains all the analysis generated by ZEUSg. There are several tabs under the **ANALYSIS** tab.

The first tab under the **ANALYSIS** tab shows the raw data extracted from the input file and is titled **Basic**. There are two data tables presented in this tab. One data table shows all the lines of data read from the input file. And the second data table shows all the words extracted from the input file:

The screenshot shows the ZEUSg Emotion Analyser web application. At the top, there's a banner with a person on a horse and the text "ZEUSg Emotion Analyser". Below it, the developer information is listed: "Developed by: Partha Majumdar" and "Riyadh (Saudi Arabia), 11-November-2019".

The main area has tabs for "Basic Information", "Statistics", "Sentiment Analysis", "Word Analysis", "Bi-Gram", and "Tri-Gram". The "Basic Information" tab is active.

**Raw Data Extracted:**

	text	Search:
1.	PARTHA MAJUMDAR	
2.	Fiat SF 05 Bangalore 56C India	legar
3.	partha	+LinkedIn: https://in.linkedin.com/in/parthmajundar6389
4.	Blog http://parthmajundar.com http://parthraevangeliqueblog.com	
5.	SUMMARY	
6.	Partha started his career in 1989 as a programmer in his first assignment he was involved in development of a Cricket Tournament management system as a part of the team from Centre for Development of Telematics CDOT requested by the Prime Minister of India Mr Rajiv Gandhi Since then Partha has developed Tea Garden automation solution Hospital Management solution Travel Management solution Manufacturing Resource Planning MRP II solution Insurance Management solution and Tax automation solution for Government of Thailand	
7.	Partha got involved in Telecom solution with project from Total Access Communications Bangkok in 1998 Partha developed the completed solution architecture and designed developed the complete infrastructure services and primitives on top of which the end-end Customer Care and Billing solution was	

**Words Extracted:**

	word	Search:
1.	partha	
2.	majundar	
3.	flat	
4.	sf	
5.	09	
6.	sneha	
7.	l	
8.	apartments	
9.	no	
10.	25	

Table 2, Records in Page: Previous 1 2 3 4 5 ... 125 Next

**Figure 9.2: Basic Information Tab of ZEUSg**

**Statistics** is the next tab under the **ANALYSIS** tab:

The screenshot shows the ZEUSg Emotion Analyser interface with the "Statistics" tab active. The top banner and developer information are the same as in Figure 9.2.

The "Basic Information" tab is still present at the top. Below it, the "Statistics" tab is active.

**Statistics Summary:**

Lines in File:	Lines with Text in File:	Unique Words in File:	Words Processed for Analysis:	Number of Words with Positive Sentiment:	Maximum/Minimum/Mean Positive Word Count:
360	233	1241	1100	132	45 1 1

**Prevalent Emotion in Text:** joy

**Distribution of Word Count:** A histogram showing the distribution of word counts. The x-axis is labeled "Count" and ranges from 0 to 40. The distribution is highly skewed to the right, with most words having a count of 1 or 2.

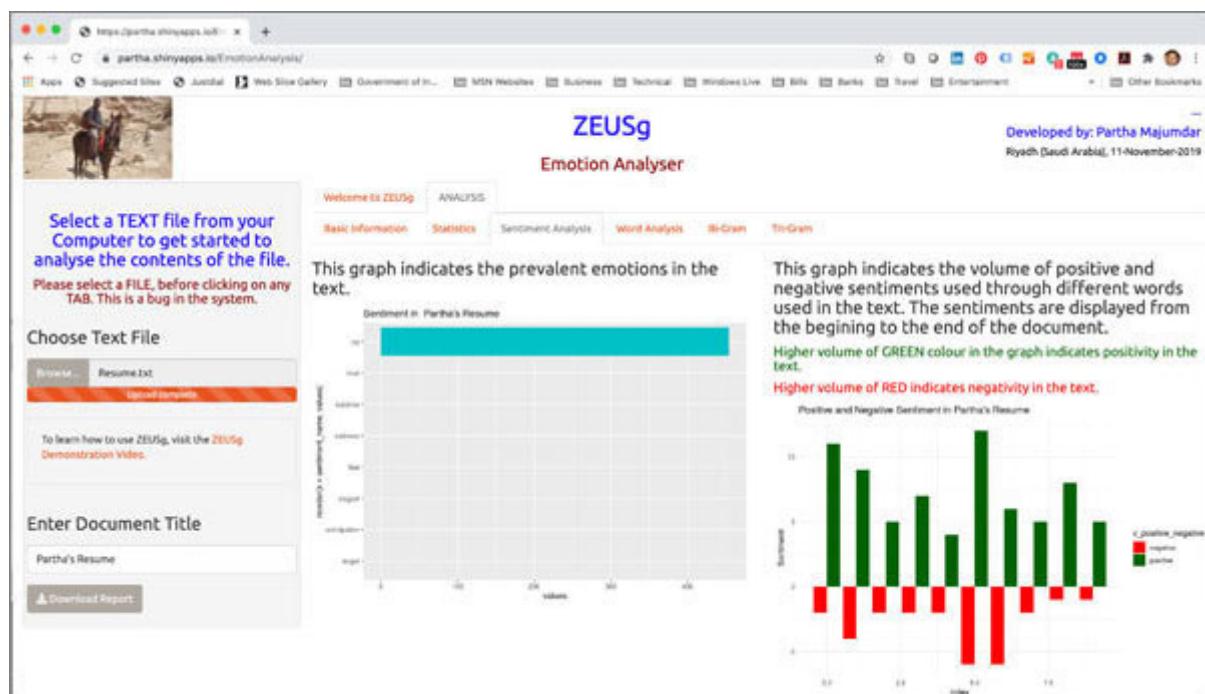
**Other Metrics:**

Total BIGRAMS in Text:	BIGRAMS processed for Analysis:
3043	1350

Total TRIGRAMS in Text:	TRIGRAMS processed for Analysis:
3617	1015

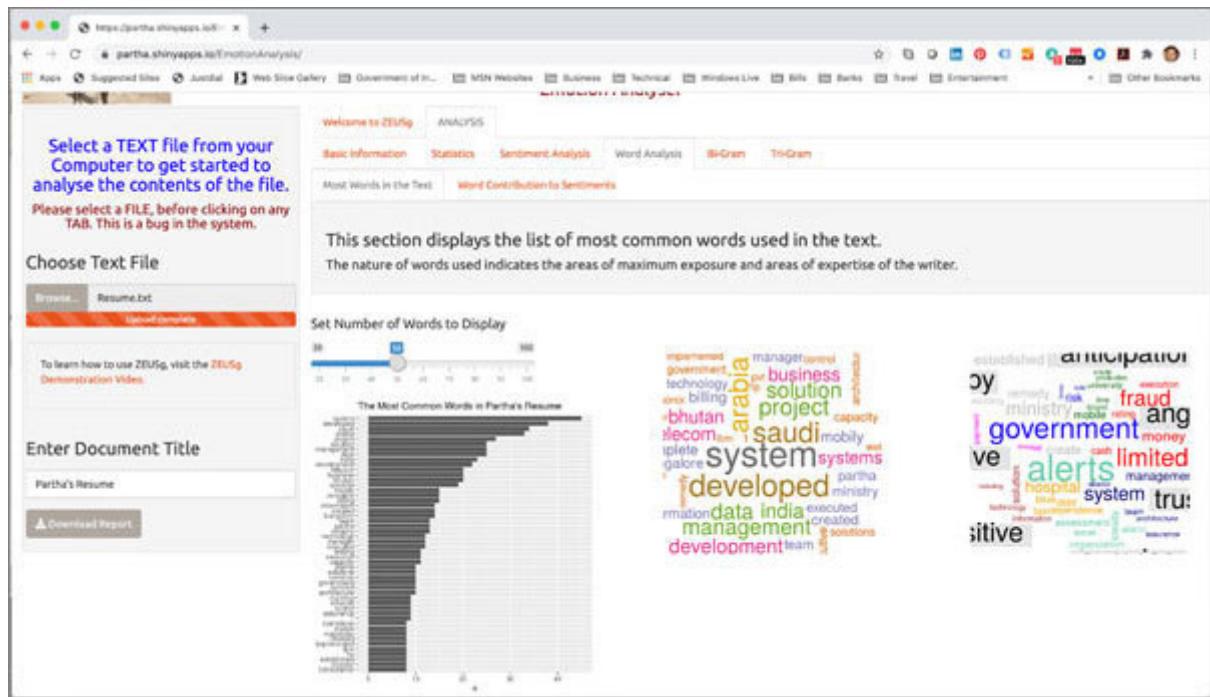
**Figure 9.3: Statistics Tab in ZEUSg**

**Sentiment Analysis** is the third tab under the **ANALYSIS** tab sentiment analysis. This tab displays the result of the emotion analysis on the left and the sentiment across the text on the right:



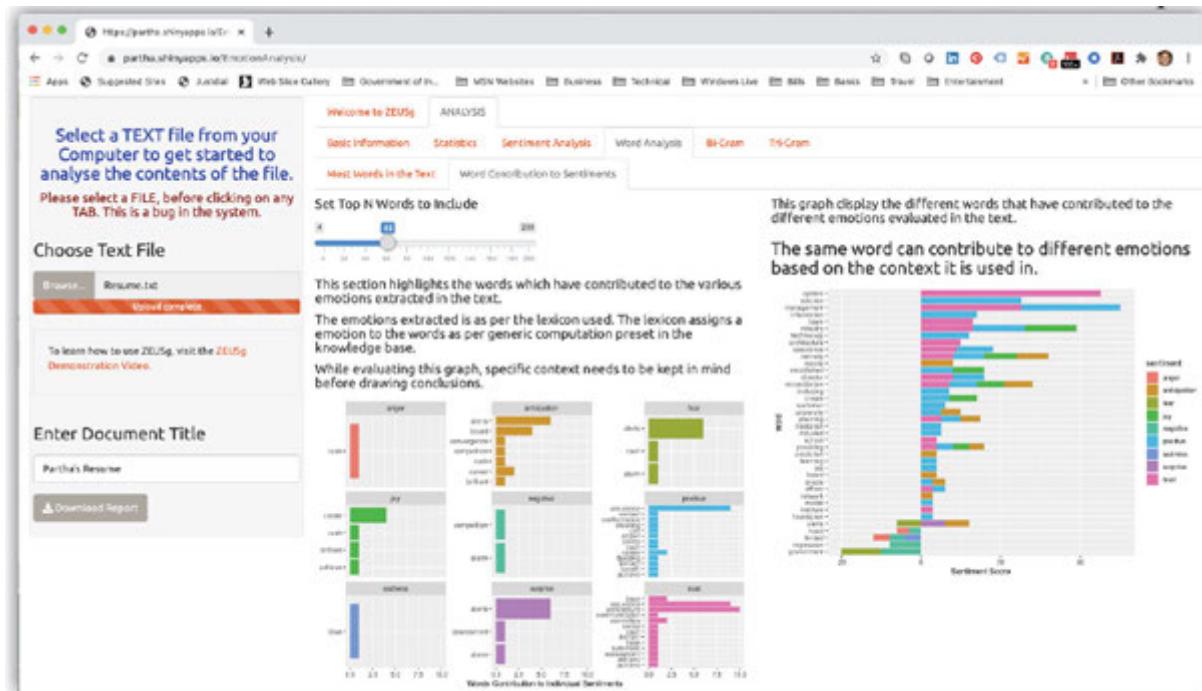
**Figure 9.4: Sentiment Analysis Tab in ZEUSg**

The fourth tab is titled word analysis. There are two tabs under this. The first tab is titled most words in the text. Under this tab, there are three visualizations<sup>3</sup> visualizations of the most words in the text. We learned how to generate these visualizations in [Chapter 8: Emotion](#). A slider is provided on this tab using which the number of words to display can be controlled:



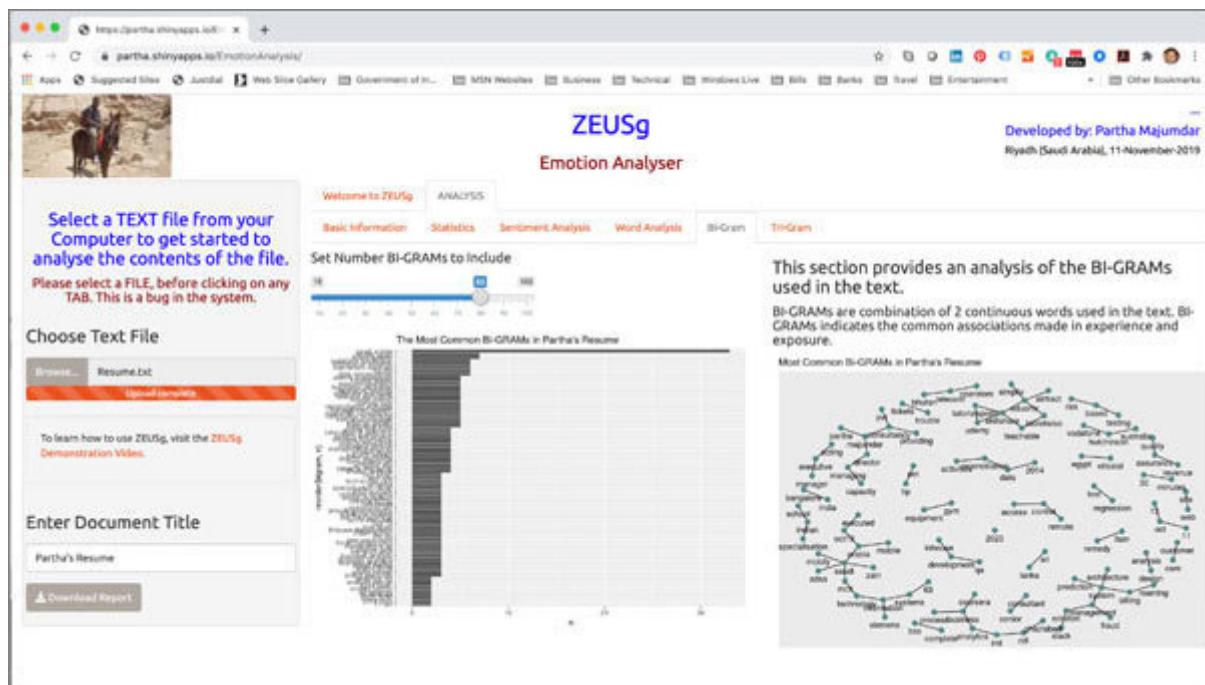
**Figure 9.5:** Most Words in the Text tab under the Word Analysis Tab in ZEUSg

The second tab under word analysis tab is word contribution to sentiments. A slider is provided here as well to control the number of words to include in the display:



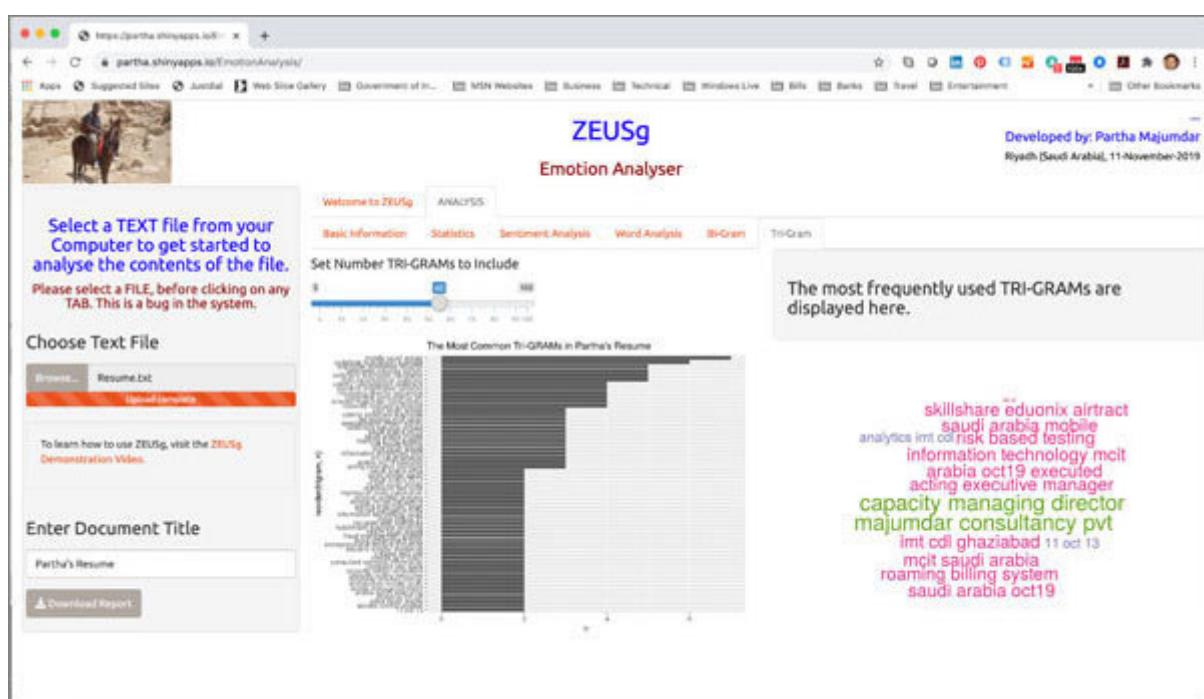
**Figure 9.6:** Word Contribution to Sentiments tab under the Word Analysis tab in ZEUSg

The fifth tab under the **ANALYSIS** tab shows the bigrams and is titled



**Figure 9.7: Bi-Gram Tab in ZEUSg**

The last tab under the **ANALYSIS** tab shows the trigrams and is titled



**Figure 9.8: Tri-Gram Tab in ZEUSg**

## [Programming ZEUSg](#)

Now, we will discuss the code needed to create ZEUSg. We learned about Shiny programming in [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#). We will use many of the concepts discussed in these chapters in developing ZEUSg and thus, I will not repeat those discussions.

## Giving credit for using a library

We use many libraries for developing an application. A good practice is to give credits to the authors of the library. We can use the citation() function to give credits for using a library shown as follows:

```
citation("textdata")
##
## To cite package 'textdata' in publications use:
##
## Emil Hvitfeldt (2020). textdata: Download and Load Various
Text
## Datasets. R package version 0.4.1.
## https://CRAN.R-project.org/package=textdata
##
## A BibTeX entry for LaTeX users is
##
## @Manual{,
##   title = {textdata: Download and Load Various Text
Datasets},
##   author = {Emil Hvitfeldt},
##   year = {2020},
##   note = {R package version 0.4.1},
##   url = {https://CRAN.R-project.org/package=textdata},
## }
```

## Libraries used to develop ZEUSg

The following is the list of libraries used to develop ZEUSg:

```
library(shiny)
library(tidytext)
library(textdata)
library(stringr)
library(dplyr)
library(tidyr)
library(ggplot2)
library(ggthemes)
library(reshape2)
library(wordcloud)
library(igraph)
library(ggraph)
library(readr)
library(textclean)
library(shinythemes)
library(DT)
```

## [Creating tabs within a tab](#)

We discussed all the elements of the ZEUSg UI in [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#). Possibly, the only new aspect is that ZEUSg has tabs within a tab. I say *possibly* because this is not really a new aspect as creating tabs within a tab is the same as creating tabs in the first place. However, let us examine this code:

```
column(width = 9,  
fluidRow(  
  tabsetPanel(id = "mainTabset", type = "tabs",  
    tabPanel("Welcome to ZEUSg",  
      img(src="CourseLogo.jpg", width=1200, height=900)  
    ),  
    tabPanel("ANALYSIS",  
      tabsetPanel(id="analysisPanel", type = "tabs",  
        tabPanel("Basic Information",  
          fluidRow(  
            column(width = 6,  
              h3("Raw Data Extracted"),  
              dataTableOutput("contents")  
            ),  
            column(width = 6,  
              h3("Words Extracted"),  
              dataTableOutput("words")  
            )  
          )  
        )  
      )  
    )  
)
```

),

This is *a* part of the code. We can see tabs in the Tabset panel mainTabset tabs. And within the tab we have declared another Tabset panel named

## Reading the input file

All the action in ZEUSg takes place once the input file is provided. So, if we discuss the events when the input file is read, we would understand the complete code of ZEUSg. There are several actions performed by ZEUSg. However, we have discussed all these actions in [Chapter 8: Emotion](#). So, I will discuss one activity and its related code, and the rest should be clear to you.

All the output controls are populated when the input file is read. One of the output controls is the data table, which data table shows the raw data in the **Basic Information** tab under the **ANALYSIS** tab. The following code populates this data table:

```
output$contents <- renderDataTable({  
  # input$file1 will be NULL initially. After the user selects  
  # and uploads a file, all rows will be shown.  
  req(input$v_input_file)  
  
  tryCatch(  
  {  
    v_text1 <- reactive({  
      tibble(text = gsub("[^[:alnum:]]///'", "",  
             read_lines(input$v_input_file$datapath)))  
    })  
  
    v_text <- reactive({
```

```
drop_empty_row(v_text1())
})
},
error = function(e) {
# return a safeError

stop(safeError(e))
}

# Extract WORDS from the Text
v_tidy_data1 <- reactive({
v_text() %>%
unnest_tokens(word, text)
})

v_tidy_data <- reactive({
v_tidy_data1() %>%
anti_join(stop_words)
})

#### Gather Statistics and Save Statistics
v_record <- gatherFileStatistics(input$v_input_file$name)
if(file.exists("./ZEUSg_statistics.csv")) {
v_statistics1 <- read.csv("./ZEUSg_statistics.csv")
v_statistics1$X <- NULL
v_statistics <- rbind(v_statistics1, v_record)
} else {
v_statistics <- v_record
}
write.csv(v_statistics, file = "./ZEUSg_statistics.csv")
```

```

DT::datatable(
  {v_text()},
  caption = htmltools::tags$caption(
    style = 'caption-side: bottom; text-align: center;',
    'Table 1: ', htmltools::em('Raw Data.'))
),

extensions = 'Buttons',

options = list(
  fixedColumns = TRUE,
  autoWidth = TRUE,
  ordering = TRUE,
  dom = 'Bftsp',
  buttons = c('copy', 'csv', 'excel')
))
)
}

```

We have discussed about most of this code in [Chapter 8: Emotion](#). The significant aspect to observe here is the use of the reactive() function. We have discussed the reactive() function in [Chapter 6: Shiny Application](#). Using the reactive() function ensures that all the involved variables are immediately refreshed when there is a change in any of the input controls. However, this also means that we have to make all the functions in the Shiny application use the reactive() function. If you notice the complete code complete code of ZEUSg, you will see that all the global functions and local functions (on the server-side) use the reactive() function. This is the only difference between the code snippets in [Chapter 8: Emotion Analysis](#) and the code of the Shiny application ZEUSg.

I could have written the code of ZEUSg better by using the feature of the reactive() function, which allows for declaring a code block within a reactive() statement. For example, look at the following code:

```
v_text1 <- reactive({  
  
  tibble(text = gsub("[^[:alnum:]]//]", "",  
         read_lines(input$v_input_file$datapath)))  
})  
  
v_text <- reactive({  
  drop_empty_row(v_text1())  
})
```

This code can be written as follows to make it more compact:

```
v_text <- reactive({  
  v_temp <- tibble(text = gsub("[^[:alnum:]]//]", "",  
                    read_lines(input$v_input_file$datapath)))  
  drop_empty_row(v_temp)  
})
```

Another new aspect in the code is the use of the tryCatch() function. Look at the following code::

```
tryCatch(  
{  
  v_text1 <- reactive({  
    tibble(text = gsub("[^[:alnum:]]//]", "",  
          read_lines(input$v_input_file$datapath)))  
  })  
  v_text <- reactive({
```

```
drop_empty_row(v_text1())
})
},
error = function(e) {
# return a safeError
stop(safeError(e))
}

)
```

What this code does is that if there is any error encountered while executing the statements in the tryCatch block, the control shifts to the error block. In the case of this code, if any error is encountered in reading the input file, ZEUSg stops executing and quits the application.

### The complete code of ZEUSg

The complete code of ZEUSg is given as follows. There are two files which comprise ZEUSg. One is app.R which contains the Shiny application. And the other is the RMarkdown

## app.R

```
citation("textdata")
```

```
##### LOAD THE LIBRARIES
```

```
#####
```

```
library(shiny)
```

```
library(tidytext)
```

```
library(textdata)
```

```
library(stringr)
```

```
library(dplyr)
```

```
library(tidyr)
```

```
library(ggplot2)
```

```
library(ggthemes)
```

```
library(reshape2)
```

```
library(wordcloud)
```

```
library(igraph)
```

```
library(ggraph)
```

```
library(readr)
```

```
library(textclean)
```

```
library(shinythemes)
```

```
library(DT)
```

```
##### INITIALISE #####
```

```
#####
```

```
v_nrc <- reactive({
```

```
get_sentiments("nrc")
```

```
)
```

```
v_positive_emotions <- reactive({  
  
  c("positive","joy","anticipation","surprise", "trust")  
})  
v_negative_emotions <- reactive({  
  c("negative","anger","disgust","fear", "sadness")  
})  
v_positive_sentiment <- reactive({  
  v_nrc() %>%  
  filter(sentiment %in% v_positive_emotions())  
})  
tryCatch(  
{  
  v_text1 <- reactive({  
    tibble(text = gsub("[^[:alnum:]]//'", "", read_lines("./Resume.txt")))  
  })  
  v_text <- reactive({  
    drop_empty_row(v_text1())  
  })  
},  
error = function(e) {  
  # return a safeError  
  stop(safeError(e))  
}  
)  
v_tidy_data1 <- reactive({  
  v_text() %>%  
  unnest_tokens(word, text)  
})  
v_tidy_data <- reactive({
```

```
v_tidy_data1() %>%
anti_join(stop_words)
})
v_unique_words <- reactive({
v_tidy_data() %>%
semi_join(v_positive_sentiment()) %>%
count(word, sort = TRUE) %>%
mutate(word = reorder(word, n))
})
emotionCount <- function() {
v_text_sentiment1 <- reactive({
v_tidy_data() %>%
inner_join(v_nrc()) %>%
count(word, index = row_number() %/% 80, sentiment) %>%
mutate(v_original_n = n) %>%
mutate(v_positive_negative = ifelse(sentiment %in%
v_positive_emotions(), 'positive', 'negative')) %>%
mutate(v_original_sentiment = sentiment) %>%
spread(sentiment, n, fill = 0)
})
v_text_sentiment <- reactive({
v_text_sentiment1() %>%
mutate(positive = ifelse("positive" %in%
colnames(v_text_sentiment1()), positive, 0)) %>%
mutate(joy = ifelse("joy" %in% colnames(v_text_sentiment1()), joy,
0)) %>%
mutate(surprise = ifelse("surprise" %in%
colnames(v_text_sentiment1()), surprise, 0)) %>%
mutate(trust = ifelse("trust" %in% colnames(v_text_sentiment1()),
trust, 0)) %>%
```

```

    mutate(anticipation = ifelse("anticipation" %in%
      colnames(v_text_sentiment1()), anticipation, 0)) %>%
    mutate(negative = ifelse("negative" %in%
      colnames(v_text_sentiment1()), negative, 0)) %>%
    mutate(anger = ifelse("anger" %in% colnames(v_text_sentiment1())),
      anger, 0)) %>%
    mutate(disgust = ifelse("disgust" %in%
      colnames(v_text_sentiment1()), disgust, 0)) %>%
    mutate(sadness = ifelse("sadness" %in%
      colnames(v_text_sentiment1()), sadness, 0)) %>%
    mutate(fear = ifelse("fear" %in% colnames(v_text_sentiment1()),
      fear, 0)) %>%
    mutate(sentiment = (positive+joy+surprise+trust+anticipation) -
      (negative+anger+disgust+sadness+fear)) %>%
    ungroup()
  })
  return(v_text_sentiment())
}
extractBiGrams <- function() {
  v_text_bigrams <- reactive({
    v_text() %>%
    unnest_tokens(bigram, text, token = "ngrams", n = 2)
  })
  return(v_text_bigrams())
}
extractBiGramsToProcess <- function() {

  v_bigrams_separated <- reactive({
    extractBiGrams() %>%
    mutate(v_original_bigram = bigram) %>%
    separate(bigram, c("word1", "word2"), sep = " ")
  })
}

```

```

v_bigrams_filtered <- reactive({
  v_bigrams_separated() %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)
})
return(v_bigrams_filtered())
}

extractTriGrams <- function() {
  v_text_trigrams <- reactive({
    v_text() %>%
    unnest_tokens(trigram, text, token = "ngrams", n = 3)
  })
  return(v_text_trigrams())
}

extractTriGramsToProcess <- function() {
  v_trigrams_separated <- reactive({
    extractTriGrams() %>%
    mutate(v_original_trigram = trigram) %>%
    separate(trigram, c("word1", "word2", "word3"), sep = " ")
  })
  v_trigrams_filtered <- reactive({
    v_trigrams_separated() %>%
    filter(!word1 %in% stop_words$word) %>%
    filter(!word2 %in% stop_words$word) %>%
    filter(!word3 %in% stop_words$word)
  })
  return(v_trigrams_filtered())
}

gatherFileStatistics <- function(p_input_file_name) {
  v_file_name <- p_input_file_name
}

```

```
v_total_lines <- nrow(v_text1())
v_lines_with_text <- nrow(v_text())
v_number_of_words <- nrow(v_tidy_data1())
v_number_of_unique_words <- nrow(unique(v_tidy_data1())))
v_words_processed <- nrow(v_tidy_data())
v_unique_words_processed <- nrow(unique(v_tidy_data()))
```

```
v_word_count <- v_tidy_data() %>% count(word)
```

```
v_word_max_frequency <- max(v_word_count$n)
v_word_min_frequency <- min(v_word_count$n)
v_word_mean_frequency <- mean(v_word_count$n)
v_word_median_frequency <- median(v_word_count$n)
```

```
v_bigram <- extractBiGrams()
v_bigram_count <- v_bigram %>% count(bigram)
```

```
v_number_of_bigrams <- nrow(v_bigram)
v_bigram_max_frequency <- max(v_bigram_count$n)
v_bigram_min_frequency <- min(v_bigram_count$n)
v_bigram_mean_frequency <- mean(v_bigram_count$n)
v_bigram_median_frequency <- median(v_bigram_count$n)
```

```
v_processed_bigram <- extractBiGramsToProcess()
```

```
v_processed_bigram_count <- v_processed_bigram %>%
count(v_original_bigram)
```

```
v_number_of_processed_bigrams <- nrow(v_processed_bigram)
```

```
v_processed_bigram_max_frequency <-  
max(v_processed_bigram_count$n)  
v_processed_bigram_min_frequency <-  
min(v_processed_bigram_count$n)  
v_processed_bigram_mean_frequency <-  
mean(v_processed_bigram_count$n)  
v_processed_bigram_median_frequency <-  
median(v_processed_bigram_count$n)
```

```
v_trigram <- extractTriGrams()  
v_trigram_count <- v_trigram %>% count(trigram)
```

```
v_number_of_trigrams <- nrow(v_trigram)  
v_trigram_max_frequency <- max(v_trigram_count$n)  
v_trigram_min_frequency <- min(v_trigram_count$n)  
v_trigram_mean_frequency <- mean(v_trigram_count$n)  
v_trigram_median_frequency <- median(v_trigram_count$n)
```

```
v_processed_trigram <- extractTriGramsToProcess()  
v_processed_trigram_count <- v_processed_trigram %>%  
count(v_original_trigram)
```

```
v_number_of_processed_trigrams <- nrow(v_processed_trigram)  
v_processed_trigram_max_frequency <-  
max(v_processed_trigram_count$n)
```

```
v_processed_trigram_min_frequency <-  
min(v_processed_trigram_count$n)  
v_processed_trigram_mean_frequency <-  
mean(v_processed_trigram_count$n)
```

```

v_processed_trigram_median_frequency <-
median(v_processed_trigram_count$n)

### Extract most prevalent Emotion
v_nrc <- emotionCount() %>%
mutate(v_original_n = ifelse(v_positive_negative == "negative", -
v_original_n, v_original_n))
v_nrc_emotions <- v_nrc %>%
select(index, anger, anticipation, disgust, fear, joy, sadness,
surprise, trust) %>%
melt(id = "index") %>%
rename(linenumber = index, sentiment_name = variable, value =
value)
v_nrc_emotions_group <- group_by(v_nrc_emotions,
sentiment_name)
v_nrc_by_emotions <- summarise(v_nrc_emotions_group,
values=sum(value))
v_nrc_temp <- v_nrc_by_emotions %>%
mutate_if(is.factor, as.character)
v_prevalent_emotion <-
ifelse(nrow(subset(v_nrc_temp, values == max(values))) == 1,
unname(unlist(subset(v_nrc_temp, values == max(values)))), 
"No Prevalent Emotion")
### Form the record
v_record <- tibble(
v_file_name,
v_total_lines,
v_lines_with_text,
v_number_of_words,
v_number_of_unique_words,
v_words_processed,

```

```
v_unique_words_processed,  
v_word_max_frequency,  
v_word_min_frequency,  
v_word_mean_frequency,  
v_word_median_frequency,  
v_number_of_bigrams,  
v_bigram_max_frequency,  
v_bigram_min_frequency,  
v_bigram_mean_frequency,  
v_bigram_median_frequency,  
v_number_of_processed_bigrams,  
v_processed_bigram_max_frequency,  
v_processed_bigram_min_frequency,  
v_processed_bigram_mean_frequency,  
v_processed_bigram_median_frequency,  
v_number_of_trigrams,  
v_trigram_max_frequency,  
v_trigram_min_frequency,  
v_trigram_mean_frequency,  
v_trigram_median_frequency,  
v_number_of_processed_trigrams,  
v_processed_trigram_max_frequency,  
v_processed_trigram_min_frequency,  
  
v_processed_trigram_mean_frequency,  
v_processed_trigram_median_frequency,  
v_prevalent_emotion  
)  
  
return(v_record)  
}
```

```
set.seed(2019)

# Define UI for application that draws a histogram
ui <- fluidPage(theme = shinytheme("united"),
tags$head(
tags$style(
HTML(".shiny-notification {
position:fixed;
top: calc(50%);
left: calc(50%);
font-family:Verdana;
fontSize:xx-large;
}
")
)
),
# Application title
fluidRow(
column(width = 3,
img(src="ParthalnPetra.jpg", width=200, height=112)
),
column(width = 6,
h1("ZEUSg", style="color:blue; text-align: center;"),
h3("Emotion Analyser", style="color:darkred; text-align: center;"),
),
column(width = 3,
h4("...", style="color:blue; text-align: right;"),
h4("Developed by: Partha Majumdar", style="color:blue; text-align: right;"),

```

```
h5("Riyadh (Saudi Arabia), 11-November-2019", style="color:black;  
text-align: right;")  
)  
,  
fluidRow(  
column(width = 3,  
wellPanel(  
fluidRow(  
h3("Select a TEXT file from your Computer to get started to  
analyze the contents of the file.", style="color:blue; text-align:  
center;"),  
h4("Please select a FILE, before clicking on any TAB. This is a  
bug in the system.", style="color:darkred; text-align: center;"),  
fileInput("v_input_file", h3("Choose Text File"),  
multiple = FALSE, placeholder = "No file selected",  
accept = c("text/csv",  
"text/comma-separated-values,text/plain",  
.txt"))  
,  
fluidRow(  
wellPanel(  
  
p("To learn how to use ZEUSg, visit the ",  
a("ZEUSg Demonstration Video.",  
href = "https://youtu.be/uEyCnHZNoMU",  
target="_blank")  
)  
)  
,  
fluidRow(  
textInput("v_document_title", h3("Enter Document Title"),  
value = "Document Title...")
```

```
),
fluidRow(
downloadButton("report", "Download Report")
)
)
),
column(width = 9,
fluidRow(
tabsetPanel(id = "mainTabset", type = "tabs",
tabPanel("Welcome to ZEUSg",
img(src="CourseLogo.jpg", width=1200, height=900)
),
tabPanel("ANALYSIS",
tabsetPanel(id="analysisPanel", type = "tabs",
tabPanel("Basic Information",
fluidRow(
column(width = 6,
h3("Raw Data Extracted"),
dataTableOutput("contents"))

),
column(width = 6,
h3("Words Extracted"),
dataTableOutput("words")
)
)
),
tabPanel("Statistics",
fluidRow(
column(width = 2,
wellPanel(
h5("Lines in File", style="color:black; text-align: center;"),
```

```
h3(textOutput("totalLines"), style="color:blue; text-align: center;")  
)  
,  
column(width = 2,  
wellPanel(  
h5("Lines with Text in File", style="color:goldenrod4; text-align:  
center;"),  
h3(textOutput("totalNonEmptyLines"), style="color:darkgreen; text-  
align: center;")  
)  
,  
column(width = 2,  
wellPanel(  
h5("Unique Words in File", style="color:goldenrod; text-align:  
center;"),
```

```
h3(textOutput("totalWords"), style="color:blue; text-align: center;")  
)  
,  
column(width = 2,  
wellPanel(  
h5("Words Processed for Analysis", style="color:darkseagreen4; text-  
align: center;"),  
h3(textOutput("totalProcessedWords"), style="color:darkgreen; text-  
align: center;")  
)  
,  
column(width = 2,  
wellPanel(  
h5("Number of Words with Positive Sentiment",  
style="color:darkmagenta; text-align: center;"),
```

```
h3(textOutput("countOfUniqueWords"), style="color:darkgreen; text-align: center;")  
)  
,  
column(width = 2,  
wellPanel(  
h5("Maximum/Minimum/Median Positive Sentiment Word Count",  
style="color:brown; text-align: center;"),  
fluidRow(  
column(width = 4,  
h3(textOutput("maxFrequencyWordCount"), style="color:cyan4; text-align: center;")  
),  
column(width = 4,  
h3(textOutput("minFrequencyWordCount"), style="color:darkturquoise; text-align: center;")  
),  
column(width = 4,  
h3(textOutput("medianFrequencyWordCount"), style="color:cyan3; text-align: center;")  
)  
)  
)  
)  
)  
,  
fluidRow(  
column(width = 4,  
fluidRow(  
column(width = 12,  
wellPanel(
```

```
h5("Prevalent Emotion in Text", style=" color:darkred; text-align: center;"),
h1(textOutput("sentiment
CountTable"), style=" color: blueviolet; text-align: center;")
)
),
),
fluidRow(
column(width = 6,
wellPanel(
h5("Total BIGRAMs in

Text", style="color:
coral3; text-align: center;"),
h1(textOutput("totalBigramCount"), style="color:blue; text-align: center;")
)
),
column(width = 6,
wellPanel(
h5("BIGRAMs processed for Analysis", style="color:coral3; text-align: center;"),
h1(textOutput ("processed Bigram Count"), style="color: darkgreen;
text-align: center;")
)
),
),
fluidRow(
column(width = 6,
wellPanel(
h5("Total TRIGRAMs in Text", style=" color: chocolate4; text-align: center;),
```

```
h1(textOutput ("total Trigram Count"), style=" color:blue; text-align: center;")  
)  
,  
column(width = 6,  
wellPanel(  
h5("TRIGRAMs processed for Analysis", style=" color: chocolate4;  
text-align: center;"),  
  
h1(textOutput ("processed Trigram Count"), style="color: darkgreen;  
text-align: center;")  
)  
)  
)  
,  
column(width = 8,  
plotOutput("wordBoxPlot")  
)  
)  
,  
tabPanel(  
title = "Sentiment Analysis",  
fluidRow(  
column(width = 6,  
h3("This graph indicates the prevalent emotions in the text."),  
plotOutput("sentiments")  
)  
),  
column(width = 6,  
h3("This graph indicates the volume of positive and negative  
sentiments used through different words used in the text. The  
sentiments are displayed from the beginning to the end of the  
document."),
```

```
h4("Higher volume of GREEN color in the graph indicates  
positivity in the text.", style="color:darkgreen; text-align: left;"),  
h4("Higher volume of RED indicates negativity in the text.",  
style="color:red; text-align: left;"),  
plotOutput("positiveNegativeSentiments")  
  
)  
)  
,  
tabPanel(  
"Word Analysis",  
tabsetPanel(type = "tabs",  
tabPanel(  
"Most Words in the Text",  
fluidRow(  
column(width = 12,  
wellPanel(  
h3("This section displays the list of most common words used in  
the text."),  
h4("The nature of words used indicates the areas of maximum  
exposure and areas of expertise of the writer.")  
)  
)  
,  
fluidRow(  
column(width = 4,  
fluidRow(  
column(width = 12,  
sliderInput("v_number_of_words",  
h4("Set Number of Words to Display"),  
min = 20, max = 100,  
value = 30, step = 5, ticks = TRUE)
```

```

),
),

plotOutput("mostWords")
),
column(width = 4,
plotOutput("wordCloud", width = "100%")
),
column(width = 4,
plotOutput("comparisonCloud", width = "100%")
)
)
),
tabPanel(
"Word Contribution to Sentiments",
fluidRow(
column(width = 6,
sliderInput("v_top_n",
h4("Set Top N Words to Include"),
min = 4, max = 200,
value = 10, step = 2, ticks = TRUE),
h4("This section highlights the words which have contributed to
the various emotions extracted in the text."),
h4("The emotions extracted is as per the lexicon used. The lexicon
assigns an emotion to the words as per generic computation
preset in the knowledge base."),
h4("While evaluating this graph, specific context needs to be kept
in mind before drawing conclusions."),
plotOutput("wordContributionToSentiments")
),
column(width = 6,

```

```

h4("This graph displays the different words that have contributed
to the different emotions evaluated in the text."),
h3("The same word can contribute to different emotions based on
the context it is used in."),
plotOutput("sentimentScore")
)
)
)
)
),
tabPanel(
"Bi-Gram",
fluidRow(
column(width = 6,
sliderInput("v_bigrams_to_include",
h4("Set Number BI-GRAMs to Include"),
min = 10, max = 100,
value = 20, step = 5, ticks = TRUE),
plotOutput("bigramFrequency")
),
column(width = 6,
h3("This section provides an analysis of the BI-GRAMs used in the
text.")),

```

```

h4("BI-GRAMs are combination of 2 continuous words used in the
text. BI-GRAMs indicates the common associations made in
experience and exposure."),."),
plotOutput("bigramGraph")
)
),
),
tabPanel(

```



```

})
output$contents <- renderDataTable({
# input$file1 will be NULL initially. After the user selects
# and uploads a file, all rows will be shown.
req(input$v_input_file)
tryCatch(
{
v_text1 <- reactive({
tibble(text = gsub("[^[:alnum:]]///'", "", 
read_lines(input$v_input_file$datapath)))
})
v_text <- reactive({
drop_empty_row(v_text1())
})
},
error = function(e) {
# return a safeError
stop(safeError(e))
}
)

```

```

# Extract WORDS from the Text
v_tidy_data1 <- reactive({
v_text() %>%
unnest_tokens(word, text)
})
v_tidy_data <- reactive({
v_tidy_data1() %>%
anti_join(stop_words)
})
### Gather Statistics and Save Statistics
v_record <- gatherFileStatistics(input$v_input_file$name)

```

```

if(file.exists("./ZEUSg_statistics.csv")) {
  v_statistics1 <- read.csv("./ZEUSg_statistics.csv")
  v_statistics1$X <- NULL
  v_statistics <- rbind(v_statistics1, v_record)
} else {
  v_statistics <- v_record
}
write.csv(v_statistics, file = "./ZEUSg_statistics.csv")
DT::datatable(
  {v_text()},
  caption = htmltools::tags$caption(
    style = 'caption-side: bottom; text-align: center;',
    'Table 1: ', htmltools::em('Raw Data.'))
),
extensions = 'Buttons',
options = list(
  fixedColumns = TRUE,
  autoWidth = TRUE,
  ordering = TRUE,
  dom = 'Bftsp',
  buttons = c('copy', 'csv', 'excel')
))
})
output$words <- renderDataTable({
  req(input$v_input_file)
  DT::datatable(
    {unique(v_tidy_data1())},
    caption = htmltools::tags$caption(
      style = 'caption-side: bottom; text-align: center;',
      'Table 2: ', htmltools::em('Words in Text.'))
  ),
}

```

```
extensions = 'Buttons',
options = list(
fixedColumns = TRUE,
autoWidth = FALSE,
ordering = TRUE,
dom = 'Bftsp',
buttons = c('copy', 'csv', 'excel')
))
})
output$totalLines <- renderText({
req(input$v_input_file)
return(nrow(v_text1()))
})
output$totalNonEmptyLines <- renderText({
req(input$v_input_file)
return(nrow(v_text())))
})

output$totalWords <- renderText({
req(input$v_input_file)
return(nrow(unique(v_tidy_data1())))
})
output$totalProcessedWords <- renderText({
req(input$v_input_file)
return(nrow(unique(v_tidy_data())))
})
output$countOfUniqueWords <- renderText({
req(input$v_input_file)
return(nrow(v_unique_words())))
})
output$maxFrequencyWordCount <- renderText({
req(input$v_input_file)
```

```
return(max(v_unique_words()$n))
})
output$minFrequencyWordCount <- renderText({
req(input$v_input_file)
return(min(v_unique_words()$n))
})
output$medianFrequencyWordCount <- renderText({
req(input$v_input_file)
return(median(v_unique_words()$n))
})
output$wordBoxPlot <- renderPlot({
req(input$v_input_file)
v_word_count <- reactive({
v_tidy_data() %>%
count(word, sort = TRUE)
})
boxplot(v_word_count()$n,
main = "Distribution of Word Count",
xlab = "Count",
ylab = NULL,
col = "orange",
border = "brown",
horizontal = TRUE,
notch = FALSE
)
})
output$totalBigramCount <- renderText({
req(input$v_input_file)
return(nrow(unique(extractBiGrams())))
})
output$processedBigramCount <- renderText({
```

```

req(input$v_input_file)
return(nrow(unique(extractBiGramsToProcess())))
})
output$totalTrigramCount <- renderText({
req(input$v_input_file)
return(nrow(unique(extractTriGrams())))
})
output$processedTrigramCount <- renderText({
req(input$v_input_file)
return(nrow(unique(extractTriGramsToProcess())))
})
#####
# MOST PREVALENT EMOTION

output$sentimentCountTable <- renderText({
req(input$v_input_file)
v_text_nrc <- reactive({
emotionCount() %>%
mutate(v_original_n = ifelse(v_positive_negative == "negative",
-v_original_n, v_original_n))
})
v_emotions <- reactive({
v_text_nrc() %>%
select(index, anger, anticipation, disgust, fear, joy,
sadness, surprise, trust) %>%
melt(id = "index") %>%
rename(linenumber = index, sentiment_name = variable, value =
value)
})
v_emotions_group <- reactive({group_by(v_emotions(),
sentiment_name)})
v_by_emotions <- reactive({summarise(v_emotions_group(),
values=sum(value))})

```

```

v_temp <- reactive({
  v_by_emotions() %>%
  mutate_if(is.factor, as.character)
})
return(
  ifelse(nrow(subset(v_temp(), values == max(values))) == 1,
    unname(unlist(subset(v_temp(), values == max(values)))),
    "No Prevalent Emotion")
)

#####
# MOST WORDS IN TEXT
output$mostWords <- renderPlot({
  req(input$v_input_file)
  v_data <- reactive({
    v_tidy_data() %>%
    count(word, sort = TRUE) %>%
    mutate(word = reorder(word, n))
  })
  v_data() %>%
    filter(row_number() <= input$v_number_of_words) %>%
    ggplot(aes(word, n)) +
    geom_col() +
    xlab(NULL) +
    coord_flip() +
    ggtitle(paste("The Most Common Words in",
      input$v_document_title, sep = " "))
})
#### WORD CLOUD
output$wordCloud <- renderPlot({
  req(input$v_input_file)
  v_data <- reactive({

```

```
v_tidy_data() %>%
  count(word, sort = TRUE) %>%
  mutate(word = reorder(word, n))
})
withProgress(message = 'Generating',
  detail = 'This may take a while...', value = 0, {
    wordcloud(words = v_data()$word, freq = v_data()$n, min.freq = 1,
      max.words = input$v_number_of_words,
      random.order=FALSE,
      colors=brewer.pal(8, "Dark2")
    )
  })
})
#####
##### COMPARISON CLOUD
output$comparisonCloud <- renderPlot({
  req(input$v_input_file)
  v_data <- reactive({
    v_tidy_data() %>%
      inner_join(v_nrc()) %>%
      count(word, sentiment, sort = TRUE) %>%
      acast(word ~ sentiment, value.var = "n", fill = 0)
  })
  withProgress(message = 'Generating',
    detail = 'This may take a while...', value = 0, {
      comparison.cloud(term.matrix = v_data(),
        colors = c("red","dark
green","blue","grey","magenta","brown","orange","mediumaquamarine","navy"),
        max.words = input$v_number_of_words
      )
    })
})
```

```
)  
output$wordContributionToSentiments <- renderPlot({  
req(input$v_input_file)  
  
v_data <- reactive({  
emotionCount() %>%  
filter(row_number() < input$v_top_n) %>%  
group_by(v_original_sentiment) %>%  
ungroup() %>%  
mutate(word = reorder(word, v_original_n))  
})  
ggplot(aes(word, v_original_n, fill = v_original_sentiment), data =  
v_data()) +  
geom_col(show.legend = FALSE) +  
facet_wrap(~v_original_sentiment, scales = "free_y") +  
labs(y = "Words Contribution to Individual Sentiments",  
x = NULL) +  
coord_flip()  
}  
output$positiveNegativeSentiments <- renderPlot({  
req(input$v_input_file)  
v_data <- reactive({  
emotionCount() %>%  
mutate(v_original_n = ifelse(v_positive_negative == "negative",  
-v_original_n, v_original_n))  
})  
ggplot(data = v_data(), aes(x = index, y = v_original_n, fill =  
v_positive_negative)) +  
geom_bar(stat = 'identity', position = position_dodge()) +  
theme_minimal() +  
ylab("Sentiment") +
```

```

ggtitle(paste("Positive and Negative Sentiment in",
input$v_document_title, sep = " ")) +
scale_color_manual(values = c("red", "dark green")) +
scale_fill_manual(values = c("red", "dark green"))
})
output$sentiments <- renderPlot({
req(input$v_input_file)
v_emotions <- reactive({
emotionCount() %>%
select(index, anger, anticipation, disgust, fear, joy,
sadness, surprise, trust) %>%
melt(id = "index") %>%
rename(linenumber = index, sentiment_name = variable, value =
value)
})
v_emotions_group <- reactive({group_by(v_emotions(),
sentiment_name)})
v_by_emotions <- reactive({summarise(v_emotions_group(),
values=sum(value))})
ggplot(aes(reorder(x=sentiment_name, values), y=values,
fill=sentiment_name), data = v_by_emotions()) +
geom_bar(stat = 'identity') +
ggtitle(paste('Sentiment in ', input$v_document_title, sep = " ")) +
coord_flip() +
theme(legend.position="none")
})
output$sentimentScore <- renderPlot({
req(input$v_input_file)
v_word_count <- reactive({
v_tidy_data() %>%
inner_join(v_nrc()) %>%

```

```

count(word, sentiment, sort = TRUE)
})
v_data <- reactive({
v_word_count() %>%
filter(row_number() < input$v_top_n) %>%
mutate(n = ifelse(sentiment %in% v_negative_emotions(), -n, n))
%>%
mutate(word = reorder(word, n))
})
ggplot(aes(word, n, fill = sentiment), data = v_data()) +
geom_col() +
coord_flip() +
labs(y = "Sentiment Score")
})
output$bigramFrequency <- renderPlot({
req(input$v_input_file)
v_bigrams_united <- reactive({
extractBiGramsToProcess() %>%
unite(bigram, word1, word2, sep = " ")
})
v_bigram_cnt <- reactive({
v_bigrams_united() %>%
count(bigram, sort = TRUE)
})
v_bigram_cnt() %>%
filter(row_number() < input$v_bigrams_to_include) %>%
ggplot(aes(x = reorder(bigram, n), y=n)) +
geom_bar(stat = 'identity') +
ggtitle(paste("The Most Common Bi-GRAMs in",
input$v_document_title, sep = " "))
coord_flip()
})

```

```

})
output$bigramGraph <- renderPlot({
req(input$v_input_file)
v_data <- reactive({
extractBiGramsToProcess() %>%
count(word1, word2, sort = TRUE)
})
withProgress(message = 'Generating',
detail = 'This may take a while...', value = 0, {
v_bigram_graph <- reactive({
v_data() %>%
filter(row_number() < input$v_bigrams_to_include) %>%
graph_from_data_frame()
})
ggraph(v_bigram_graph(), layout = "kk") +
geom_edge_link() +
geom_node_point(color = "darkslategray4", size = 3) +
geom_node_text(aes(label = name), vjust = 1.8) +
ggtitle(paste("Most Common Bi-GRAMs in", input$v_document_title,
sep = " "))
})
})
output$trigramFrequency <- renderPlot({
req(input$v_input_file)

v_trigrams_united <- reactive({
extractTriGramsToProcess() %>%
unite(trigram, word1, word2, word3, sep = " ")
})
v_trigram_cnt <- reactive({
v_trigrams_united() %>%
count(trigram, sort = TRUE)
})
})
```

```

})
v_trigram_cnt() %>%
filter(row_number() < input$v_trigrams_to_include) %>%
ggplot(aes(x = reorder(trigram, n), y=n)) +
geom_bar(stat = 'identity') +
ggtitle(paste("The Most Common Tri-GRAMs in",
input$v_document_title, sep = " ")) +
coord_flip()
})
output$trigramWordCloud <- renderPlot({
req(input$v_input_file)
v_trigrams_united <- reactive({
extractTriGramsToProcess() %>%
unite(trigram, word1, word2, word3, sep = " ")
})
v_trigram_cnt <- reactive({
v_trigrams_united() %>%
count(trigram, sort = TRUE)
})
withProgress(message = 'Generating',
detail = 'This may take a while...', value = 0, {
wordcloud(words = v_trigram_cnt()$trigram,
freq = v_trigram_cnt()$n,
min.freq = 1,
max.words = input$v_trigrams_to_include,
random.order=FALSE,
colors=brewer.pal(8, "Dark2"))
})
})
output$report <- downloadHandler(
filename <- "report.pdf",

```

```

content <- function(file) {
  withProgress(message = 'Generating Report...', value = 0, {
    # Set up parameters to pass to Rmd document
    params <- list(
      p_input_file = input$v_input_file$name,
      p_document_title = input$v_document_title,
      p_number_of_words = input$v_number_of_words,
      p_top_n = input$v_top_n,
      p_bigrams_to_include = input$v_bigrams_to_include,
      p_trigrams_to_include = input$v_trigrams_to_include,
      p_text = v_text(),
      p_total_lines = nrow(v_text1())
    )
    rmarkdown::render("ZEUSg.Rmd",
      output_file = file,
      params = params,
      envir = new.env(parent = globalenv())
    )
  }
}

}

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

```

## ***ZEUSg.Rmd***

```

---
title: "Emotion Analysis"
author: "Partha Majumdar"
date: "r format(Sys.Date(), "%B %d, %Y")"

```

```
output:  
pdf_document:  
fig_caption: yes  
keep_tex: yes  
number_sections: yes  
toc: yes  
html_document: default  
word_document: default  
params:  
p_input_file: NA  
p_document_title: NA  
p_number_of_words: NA  
p_top_n: NA  
p_bigrams_to_include: NA  
p_trigrams_to_include: NA  
p_text: NA  
p_total_lines: NA  
p_total_words: NA
```

---

```
'''{r load_libraries, include=FALSE, echo=FALSE, warning=FALSE}  
knitr::opts_chunk$set(echo = TRUE)
```

```
library(tidytext)  
library(textdata)  
library(stringr)  
library(dplyr)  
library(tidyr)  
library(ggplot2)  
library(ggthemes)
```

```
library(reshape2)
library(wordcloud)
library(igraph)
library(ggraph)
library(readr)
library(textclean)
library(kableExtra)
...
```
```
```{r set_variables, include=FALSE, echo=FALSE, warning=FALSE}
v_document_title <- params$p_document_title
```
```
v_nrc <- get_sentiments("nrc")
```
```
v_positive_emotions <- c("positive","joy","anticipation","surprise",
"trust")
v_negative_emotions <- c("negative","anger","disgust","fear",
"sadness")
...
```
```
```{r read_file, include=FALSE, echo=FALSE, warning=FALSE}
v_text <- params$p_text
v_text <- drop_empty_row(v_text)
...
```
```
```
```
```{r extract_base_data_for_analysis, include=FALSE, echo=FALSE,
warning=FALSE}
v_tidy_data1 <- v_text %>%
unnest_tokens(word, text)
v_tidy_data <- v_tidy_data1 %>%
anti_join(stop_words)
```

```
v_initial_counts <- v_tidy_data %>%
count(word, sort = TRUE)
...
# Analysis of the Document: 'r v_document_title'
##File Name: 'r params$p_input_file'##
##Introduction
This report is generated by **ZEUSg**, an **Emotion Analyser**. **ZEUSg** can identify the positive and negative sentiments expressed in the document. The positive and negative sentiments are based on the analysis of the words in the document which are matched against the knowledge base available in**ZEUSg**. **ZEUSg** can match words against 10 different emotions. The emotions are **positive**, **joy**, **anticipation**, **trust**, **surprise**, **negative**, **anger**, **sadness**, **disgust** and **fear**. The emotions *positive*, *joy*, *anticipation*, *trust* and *surprise* are the **POSITIVE Sentiments**. While the emotions *negative*, *anger*, *sadness*, *disgust* and *fear* are the **NEGATIVE Sentiments**.
```

\*\*ZEUSg\*\* evaluates the words for the different emotions based on the context in which the word is used \*in a limited way\*. While making conclusions from the report, it is important to consider putting the findings in the context that the analysis is being applied to. This aspect has to be conducted through Human Expertise.

We have applied \*\*ZEUSg\*\* in \*\*Human Resource Departments\*\* for analysing \*Resumes of candidates\* during the Screening Process. \*\*ZEUSg\*\* provides an insight into the key contents of a Resume and identifies the main emotion(s) expressed in the Resume. It highlights the main words, bi-grams and tri-grams used in the Resume. Further, it reports how each of the words are analyzed for different Emotions in the Contexts extracted from

the Resume. At the very minimum, it provides the key information at a glance which helps in making decisions.

We are also using \*\*ZEUSg\*\* in analysing \*\*Interactions with the Customers and Vendors through Emails\*\*. The contents of the Emails are stored in a \*\*TEXT\*\* file and processed through \*\*ZEUSg\*\*. Apart from the other analysis as discussed previously, \*\*ZEUSg\*\* provides insight into how the interactions have been progressing over time in terms of \*being positive\* or \*being negative\*. This helps in changing strategies for further interactions and/or taking suitable actions for the best results.

This application of \*\*ZEUSg\*\* can be extended to conversations over phone call or through any other medium. The conversation recordings can be converted to TEXT using suitable software and the transcripts can be analyzed using \*\*ZEUSg\*\*.

\*\*ZEUSg\*\* can be used to analyze \*\*books\*\*, \*\*technical/non-technical documents\*\* and the list is endless.

## ##About this Report

The report states the findings from the document. However, the report does not draw any conclusions. \*\*The reader of the report has to draw the conclusions based on the facts provided in the report.\*\*

\pagebreak

```
### Summary of the Finding in the Document: **'r  
params$p_input_file'**
```

```
*Total Number of Lines in the Document* = **'r  
params$p_total_lines'**
```

```
*Number of Lines with Text in the Document* = **'r  
nrow(v_text)'**
```

```
*Total Number of Words in the Document* = **'r  
nrow(v_tidy_data1)'**
```

```
*Number of Unique Words in the Document* = **'r  
nrow(unique(v_tidy_data1))'**  
*Number of Words considered for Processing* = **'r  
nrow(v_tidy_data)'**  
*Number of Unique Words considered for Processing* = **'r  
nrow(unique(v_tidy_data))'**  
*Word with Maximum Frequency* = **'r  
head(subset(v_initial_counts, n == max(n))$word, 1)' ('r  
max(v_initial_counts$n))'**  
*Minimum Frequency of ONE Words* = **'r  
min(v_initial_counts$n)'**  
*Mean Frequency of Words* = **'r mean(v_initial_counts$n)'**  
*Median Frequency of Words* = **'r median(v_initial_counts$n)'**  
""'{r box_plot, echo=FALSE, include=TRUE, warning=FALSE}
```

```
boxplot(v_initial_counts$n,  
main = "Distribution of Word Count",  
xlab = "Count",  
ylab = NULL,  
col = "orange",  
border = "brown",  
horizontal = TRUE,  
notch = FALSE  
)  
...  
"
```

```
""'{r histogram, echo=FALSE, include=TRUE, warning=FALSE}  
hist(v_initial_counts$n,  
main = "Histogram of Word Count"  
)  
..."
```

```
\pagebreak  
# MOST COMMON WORDS IN THE TEXT
```

This section displays the list of most common words used in the text. The nature of words used indicates the areas of maximum exposure & areas of expertise and/or prevalent emotional state while writing the text.

The below graphs include \*\*'r params\$p\_number\_of\_words'\*\* most frequently occurring words.

```
'''{r most_words_in_text, echo=FALSE, include=FALSE,  
warning=FALSE}  
v_positive_sentiment <- v_nrc %>% # get_sentiments(v_lexicon)  
%>%  
filter(sentiment %in% v_positive_emotions)
```

```
v_tidy_data %>%  
semi_join(v_positive_sentiment) %>%  
count(word, sort = TRUE)  
'''
```

```
'''{r most_words_in_text_display, echo=FALSE, include=TRUE,  
fig.align="center", warning=FALSE}  
v_tidy_data %>%  
count(word, sort = TRUE) %>%  
filter(row_number() <= params$p_number_of_words) %>%  
mutate(word = reorder(word, n)) %>%  
ggplot(aes(word, n)) +
```

```
geom_col() +  
xlab(NULL) +  
coord_flip() +  
ggtitle(paste("The Most Common Words in", v_document_title, sep  
= " "))  
...  
  
...
```

```
\pagebreak  
## WORD CLOUD
```

The Word Cloud displays the \*\*'r params\$p\_number\_of\_words'\*\* most used words in the text.

```
""{r word_cloud_display, echo=FALSE, include=TRUE,  
fig.align="center", warning=FALSE}  
v_data <- v_tidy_data %>%  
count(word, sort = TRUE) %>%  
mutate(word = reorder(word, n))  
  
wordcloud(words = v_data$word, freq = v_data$n, min.freq = 1,
```

```
max.words = params$p_number_of_words,  
random.order=FALSE,  
# rot.per=0.35,  
colors=brewer.pal(8, "Dark2")  
)  
...  
  
...
```

```
\pagebreak  
## EMOTION WORD CLOUD
```

The Word Cloud displays the \*\*`r params\$p\_number\_of\_words`\*\* most used words in the text.

The size of the words indicates the frequency of the use of the individual words. The larger the size of the word, the more frequently the word has been used.

The colors indicate the different emotions in which context the words have been evaluated by the program.

```
'''{r emotion_cloud_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
v_tidy_data %>%
inner_join(v_nrc) %>%
count(word, sentiment, sort = TRUE) %>%
acast(word ~ sentiment, value.var = "n", fill = 0) %>%
comparison.cloud(
colors = c("red", "dark green", "blue", "grey", "magenta", "brown",
"orange", "mediumaquamarine", "navy"),
max.words = params$p_number_of_words)
'''
```

\pagebreak

## # WORD CONTRIBUTION TO SENTIMENTS

This section highlights the words which have contributed to the various emotions extracted from the text. The emotions extracted are as per the lexicon used. The lexicon assigns an emotion to the words as per generic computation preset in the knowledge base. \*\*While evaluating this graph, specific context needs to be kept in mind before drawing conclusions.\*\*

```
'''{r word_contribution_to_sentiments, echo=FALSE, include=FALSE,
warning=FALSE}
v_text_sentiment <- v_tidy_data %>%
inner_join(v_nrc) %>%
count(word,
index = row_number() %/%
80, sentiment) %>%
mutate(v_original_n = n) %>%
mutate(v_positive_negative = ifelse(sentiment %in%
v_positive_emotions, 'positive', 'negative')) %>%
mutate(v_original_sentiment = sentiment) %>%
spread(sentiment, n, fill = 0)

v_text_sentiment <- v_text_sentiment %>%
mutate(positive = ifelse("positive" %in%
colnames(v_text_sentiment), positive, 0)) %>%
mutate(joy = ifelse("joy" %in% colnames(v_text_sentiment), joy, 0))
%>%
mutate(surprise = ifelse("surprise" %in%
colnames(v_text_sentiment), surprise, 0)) %>%
mutate(trust = ifelse("trust" %in% colnames(v_text_sentiment),
trust, 0)) %>%
mutate(anticipation = ifelse("anticipation" %in%
colnames(v_text_sentiment), anticipation, 0)) %>%

mutate(negative = ifelse("negative" %in%
colnames(v_text_sentiment), negative, 0)) %>%
mutate(anger = ifelse("anger" %in% colnames(v_text_sentiment),
anger, 0)) %>%
mutate(disgust = ifelse("disgust" %in% colnames(v_text_sentiment),
disgust, 0)) %>%
mutate(sadness = ifelse("sadness" %in%
colnames(v_text_sentiment), sadness, 0)) %>%
```

```

mutate(fear = ifelse("fear" %in% colnames(v_text_sentiment), fear,
o)) %>%
mutate(sentiment = (positive+joy+surprise+trust+anticipation) -
(negative+anger+disgust+sadness+fear)) %>%
ungroup()
...
```
```
""{r word_contribution_to_sentiments_display, echo=FALSE,
include=TRUE, fig.align="center", warning=FALSE}
v_text_sentiment %>%
filter(row_number() < params$p_top_n) %>%
group_by(v_original_sentiment) %>%
ungroup() %>%
mutate(word = reorder(word, v_original_n)) %>%
ggplot(aes(word, v_original_n, fill = v_original_sentiment)) +
geom_col(show.legend = FALSE) +
facet_wrap(~v_original_sentiment, scales = "free_y") +
labs(y = "Words Contribution to Individual Sentiments",
x = NULL) +
coord_flip()
...
```

```

\pagebreak

## # POSITIVE & NEGATIVE SENTIMENTS

This graph indicates the volume of positive and negative sentiments used through different words used in the text. Higher volume of **GREEN** colour in the graph indicates positivity in the text, while a higher volume of **RED** indicates negativity in the text.

The graph displays the positive and negative Sentiments from the start of the document to the end. So, it is possible infer the Sentiments in the document as it has progressed.

```
'''{r positive_and_negative_sentiments, echo=FALSE, include=FALSE,
warning=FALSE}
v_text_nrc <- v_text_sentiment %>%
  mutate(v_original_n = ifelse(v_positive_negative == "negative", -
    v_original_n, v_original_n))
...
'''{r positive_and_negative_sentiments_display, echo=FALSE,
include=TRUE, fig.align="center", warning=FALSE}
ggplot(data = v_text_nrc, aes(x = index, y = v_original_n, fill =
v_positive_negative)) +
  geom_bar(stat = 'identity', position = position_dodge()) +
  theme_minimal() +
  ylab("Sentiment") +
  ggtitle(paste("Positive and Negative Sentiment in",
v_document_title, sep = " ")) +
  scale_color_manual(values = c("red", "dark green")) +
  scale_fill_manual(values = c("red", "dark green"))
...
\pagebreak
# SENTIMENTS
```

This graph indicates the prevalent emotions in the text.

```
'''{r sentiments, echo=FALSE, include=FALSE, warning=FALSE}
v_emotions <- v_text_nrc %>%
  select(index, anger, anticipation, disgust, fear, joy,
sadness, surprise, trust) %>%
```

```

melt(id = "index")
names(v_emotions) <- c("linenumber", "sentiment", "value")
v_emotions_group <- group_by(v_emotions, sentiment)
v_by_emotions <- summarise(v_emotions_group,
values=sum(value))
"""

"""

``{r sentiments_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
ggplot(aes(reorder(x=sentiment, values), y=values, fill=sentiment),
data = v_by_emotions) +
geom_bar(stat = 'identity') +
ggtitle(paste('Sentiment in ', v_document_title, sep = " ")) +
coord_flip() +
theme(legend.position="none")
```

\pagebreak
# SENTIMENT SCORE

```

This graph display the different words that have contributed to the different emotions evaluated in the text. \*\*The same word can contribute to different emotions based on the context it is used in.\*\*

```

``{r sentiment_score, echo=FALSE, include=FALSE, warning=FALSE}
v_word_count <- v_tidy_data %>%
inner_join(v_nrc) %>%
count(word, sentiment, sort = TRUE)
```

```

```

'''{r sentiment_score_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
v_word_count %>%
filter(row_number() < params$p_top_n) %>%
mutate(n = ifelse(sentiment %in% v_negative_emotions, -n, n))
%>%
mutate(word = reorder(word, n)) %>%
ggplot(aes(word, n, fill = sentiment))+
geom_col() +
coord_flip() +
labs(y = "Sentiment Score")
'''
```

\pagebreak

## # BI-GRAM ANALYSIS

This section provides an analysis of the **\*\*BI-GRAMs\*\*** used in the text. BI-GRAMs are combination of 2 consecutive words used in the text. BI-GRAMs indicate the common associations made in experience and exposure.

```

'''{r bigram_frequency, echo=FALSE, include=FALSE,
warning=FALSE}
```

```
v_text_bigrams <- v_text %>%
```

```
unnest_tokens(bigram, text, token = "ngrams", n = 2)
```

```
v_bigrams_separated <- v_text_bigrams %>%
separate(bigram, c("word1", "word2"), sep = " ")
```

```
v_bigrams_filtered <- v_bigrams_separated %>%
filter(!word1 %in% stop_words$word) %>%
filter(!word2 %in% stop_words$word)
```

```
v_bigram_counts <- v_bigrams_filtered %>%
count(word1, word2, sort = TRUE)
```

```
v_bigrams_united <- v_bigrams_filtered %>%
unite(bigram, word1, word2, sep = " ")
```

```
v_bigram_cnt <- v_bigrams_united %>%
count(bigram, sort = TRUE)
```

```
v_bigram_cnt <- v_bigram_cnt %>%
filter(row_number() < params$p_bigrams_to_include)
...  
  
This graph displays the most frequently used BIGRAMs.
```

```
'''{r bigram_frequency_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
ggplot(aes(x = reorder(bigram, n), y=n), data=v_bigram_cnt) +
geom_bar(stat = 'identity') +
ggttitle(paste("The Most Common Bigrams in", v_document_title,
sep = " ")) +
coord_flip()
...  
  
\pagebreak
```

```
## BI-GRAM Word Cloud
```

```
'''{r bigram_word_cloud_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
wordcloud(words = v_bigram_cnt$bigram, freq = v_bigram_cnt$n,
min.freq = 1,
max.words = params$p_bigrams_to_include,
random.order=FALSE,
#
# rot.per=0.35,
colors=brewer.pal(8, "Dark2")
)
'''

'''{r bigram_graph, echo=FALSE, include=FALSE, warning=FALSE}
v_bigram_graph <- v_bigram_counts %>%
filter(row_number() < params$p_bigrams_to_include) %>%
graph_from_data_frame()

set.seed(2019)
'''
```

\pagebreak

This graphic provides a pictorial view of the most frequently used BI-GRAMs.

```
'''{r bigram_graph_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
ggraph(v_bigram_graph, layout = "kk") +
geom_edge_link() +
geom_node_point(color = "darkslategray4", size = 3) +
geom_node_text(aes(label = name), vjust = 1.8) +
```

```
ggtile(paste("Most Common Bigrams in", v_document_title, sep =  
" "))  
...  
\\pagebreak
```

## # TRI-GRAM ANALYSIS

This section provides an analysis of the \*\*TRI-GRAMs\*\* used in the text. TRI-GRAMs are combination of 3 consecutive words used in the text. Just like BI-GRAMs, TRI-GRAMs indicate the common associations made in experience and exposure.

```
'''{r trigram_frequency, echo=FALSE, include=FALSE,  
warning=FALSE}  
v_book_trigrams <- v_text %>%  
unnest_tokens(trigram, text, token = "ngrams", n = 3)
```

```
v_trigrams_separated <- v_book_trigrams %>%  
separate(trigram, c("word1", "word2", "word3"), sep = " ")
```

```
v_trigrams_filtered <- v_trigrams_separated %>%  
filter(!word1 %in% stop_words$word) %>%  
filter(!word2 %in% stop_words$word) %>%  
filter(!word3 %in% stop_words$word)
```

```
v_trigram_counts <- v_trigrams_filtered %>%  
count(word1, word2, word3, sort = TRUE)
```

```
v_trigrams_united <- v_trigrams_filtered %>%  
unite(trigram, word1, word2, word3, sep = " ")
```

```
v_trigram_cnt <- v_trigrams_united %>%
count(trigram, sort = TRUE)
```

```
v_trigram_cnt <- v_trigram_cnt %>%
filter(row_number() < params$p_trigrams_to_include)
...  
  

```

This graph displays the most frequently used TRI-GRAMs.

```
""{r trigram_frequency_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
ggplot(aes(x = reorder(trigram, n), y=n), data=v_trigram_cnt) +
geom_bar(stat = 'identity') +
ggttitle(paste("The Most Common Trigrams in", v_document_title,
sep = " ")) +
coord_flip()
""
```

\pagebreak

## TRI-GRAM Word Cloud

```
""{r trigram_word_cloud_display, echo=FALSE, include=TRUE,
fig.align="center", warning=FALSE}
wordcloud(words = v_trigram_cnt$trigram, freq = v_trigram_cnt$n,
min.freq = 1,
max.words = params$p_trigrams_to_include,
random.order=FALSE,
# rot.per=0.35,
```

```
colors=brewer.pal(8, "Dark2")
)
...
\center
style="color:red">**----- THE END -----**
\center
```

## Conclusion

We built a decent sized application using Shiny programming in this chapter. We made use of all the concepts of Shiny that programming we learned in [Chapter 5: Shiny Application 1](#) and [Chapter 6: Shiny Application](#). The application was fully responsive using the reactive() function.

As is required for any application development, we should have a design before we start developing an application. So, we started the discussion with the design aspect of ZEUSg. As ZEUSg was an application which was already developed, we discussed the actual screens that would be developed. Normally, these would be sketches on a paper or software like Adobe XD. The important aspect is that we should always have a plan before starting on a project and for a web application, the design is the blueprint.

I shared the complete code for ZEUSg. As discussed in the chapter, this code can be optimized and beautified. So, please use all the skills that you mastered in the book to create beautiful Shiny applications.

In the next chapter, we will study another application of Emotion Analysis on data collected from Twitter.

### Points to remember

Credits can be given using the citation() function.

Tabs can be created within another tab by declaring a tablet in that tab.

Try...Catch...Finally structure can be programmed in Shiny applications using the tryCatch() function.

### Multiple choice questions

The function to write to a CSV file is:

CSV.Write

Write.csv

write.csv

None of these

Which of these functions can be used to extract Bigrams?

extract.bigram

extract.ngram

unnest\_tokens

None of these

Which of these functions can be used to extract Trigrams?

extract.trigram

`extract.ngram`

`unnest_tokens`

None of these

The function to find a Median is:

`median`

`Median`

Both of these

None of these

The function to take a file input in Shiny programming is:

`File.input`

`fileInput`

`file.input`

None of these

## Answers to MCQs

C

C

C

A

B

## Questions

Enhance ZEUSg to accept file types other than TEXT, files like PDF files, DOC files, etc. as input. What challenges do you encounter with the present code of ZEUSg when doing so?

Create an emotion analysis model for any language other than English. Create a Shiny application for the same.

## Key terms

**N-gram:** N-grams of texts are extensively used in Text Mining and Natural Language Processing tasks. They are basically a set of co-occurring words within a given window.

## SECTION - 4

## CHAPTER 10

### Introduction to Twitter Data Analysis

In [Chapter 8: Emotion Analysis](#), we discussed how to conduct emotion analysis on any given text. Now, we will move to the next data set to conduct on emotion analysis - the data obtained from Twitter. This chapter introduces Twitter and explains the enormity of data available from social networking site. We will discuss some applications developed using data derived from Twitter.

Then, we will discuss how to create a Twitter developer account and how to obtain a Google Maps API Key. Both are essential for making the R programs that programs we will discuss in [Chapter 11: Emotion Analysis on Twitter Data](#) and [Chapter 12: Chidiya](#) for emotion analysis on data obtained from Twitter.

## **Structure**

In this chapter, we will discuss the following topics:

Introduction to Twitter

Twitter Developer Account

Google Maps API Key

## Objectives

After studying this unit, you should be able to:

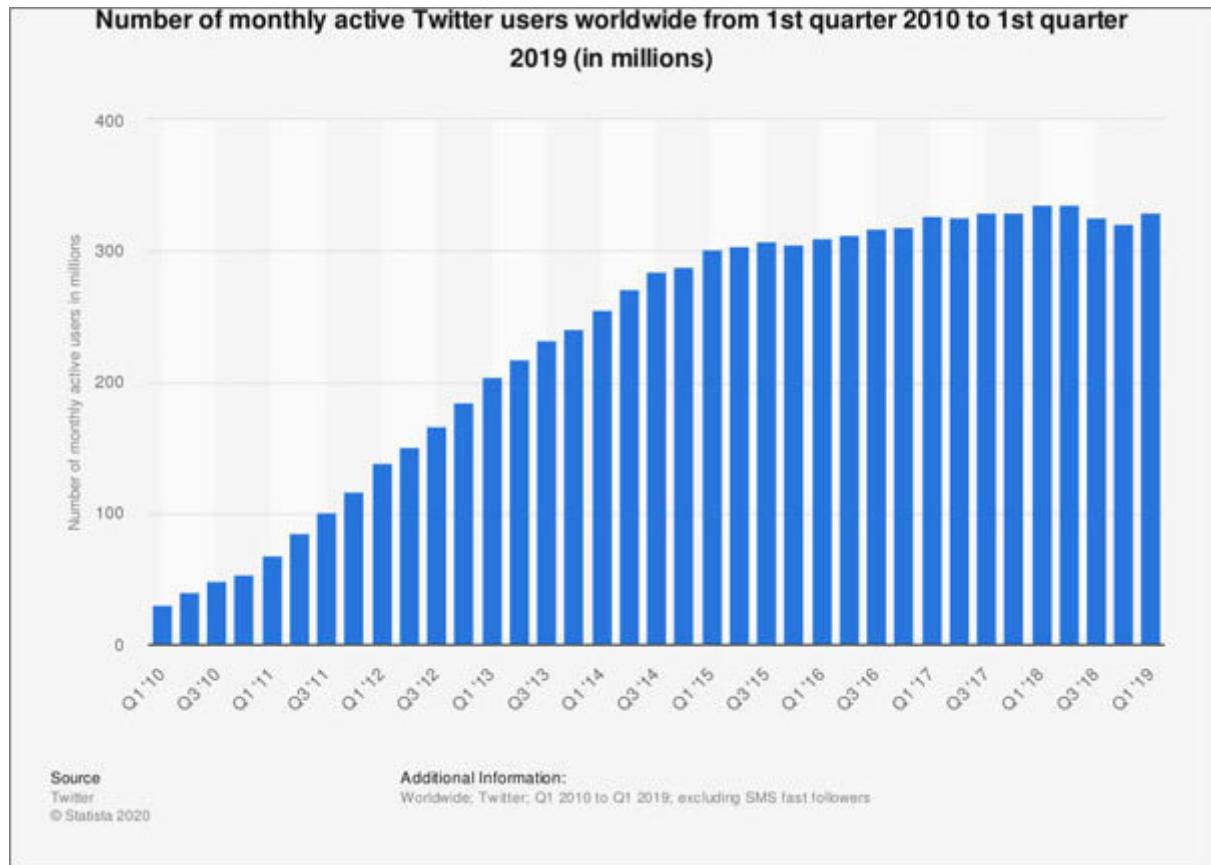
Create a Twitter Developer Account

Obtain a Google Maps API Key

## Introduction to Twitter

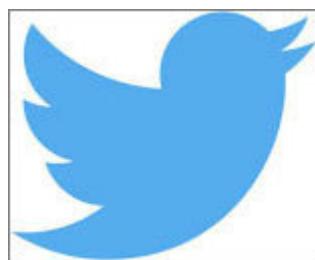
Twitter is one of the most popular Social Media platforms. It is available at Presently, there are approximately 330 million monthly active users on Twitter and approximately 145 million daily active users on Twitter. Twitter users include most of the world leaders, scientists, doctors, and celebrities, who make their point of view known to the world through Twitter. Almost all the users of Twitter make all their major (and minor) announcements on Twitter and use the platform to discuss almost all the latest events in the world

The growth of the Twitter user base can be seen in the following graph:



**Figure 10.1:** Growth of Twitter User Base between 2010 to 2019

Twitter was launched on 2006 in San Francisco, United States of America. Twitter was launched as a mobile application (more popularly called a mobile app) on 2006. Twitter is easily recognized by its logo which is shown as follows:



**Figure 10.2:** Twitter Logo

What one writes on Twitter is called a tweet. The users of Twitter can write any text in almost any language and include URLs, videos, and photographs in their tweets. One tweet can contain a maximum of 280 characters. When one writes a tweet, we say that the user has posted a tweet. Other users of Twitter can view the tweets and conduct many actions on them. For example, Twitter users can like the tweets, retweet the original tweet, reply to that tweet, etc.

Presently, approximately 500 million tweets are posted daily. The result is that there is a huge volume of data available on Twitter. Twitter makes this data available for analysis. By analyzing Twitter data, several interpretations of the daily lives can be made. For example, when a political leader announces some new policy, analyzing it is possible to find out from which parts of the world what volume of people are talking about the policy by analyzing Twitter data. It is further possible to find out the percentage of people who think positively or negatively or are neutral about the announcement percentage of people thinks.

Another popular application of Twitter data is to use it for training the computer to be able to write in a language. This field of computer science falls under **Natural Language Processing**. This is because the users of Twitter express themselves in their native ways in the tweets. So, the raw data available from Twitter is how people would normally express themselves and communicate. This is different from other such sources like newspapers, magazines or books as in these medium, the author normally curates what he/she writes. As users of Twitter post tweets in almost all languages, it is possible to teach the computer to write in any of

the languages using Twitter data. This can be extrapolated to the fact that if the computer can be made to write, it can be made to speak as well.

In this book, we will use Twitter data to extract the emotion expressed in a collection of tweets. We will use the concept of emotions analysis and the method for extracting emotions from any text using R programming as discussed in [Chapter 7: Sentiment Analysis](#) and [Chapter 8: Emotion](#)

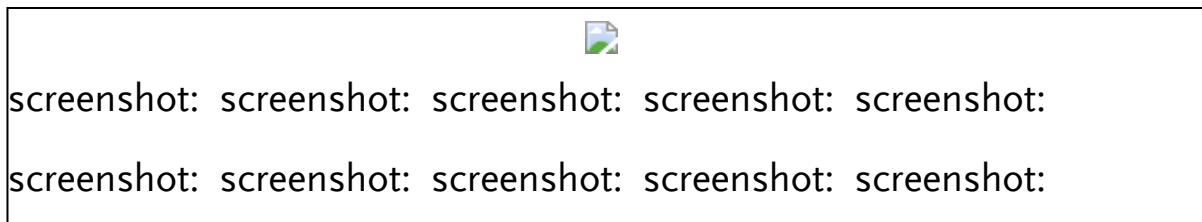
## Twitter Developer Account

Before we can get data from Twitter and conduct analysis on the same, we need to obtain a Twitter Developer Account. Let us go through the process of obtaining the same.

**We can fetch the data from** Twitter without having a Twitter Developer Account. However, we cannot be guaranteed all the data that we need from Twitter is actually fetched. Also, it is impossible to fetch data from Twitter beyond a certain limit without having a Twitter Developer Account.

## URL for creating a Twitter Developer Account

Visit the URL <https://developer.twitter.com/> to create a Twitter Developer Account. On invoking this URL from Google Chrome, you should get a UI like the one shown on the left-hand side of the following screenshot:

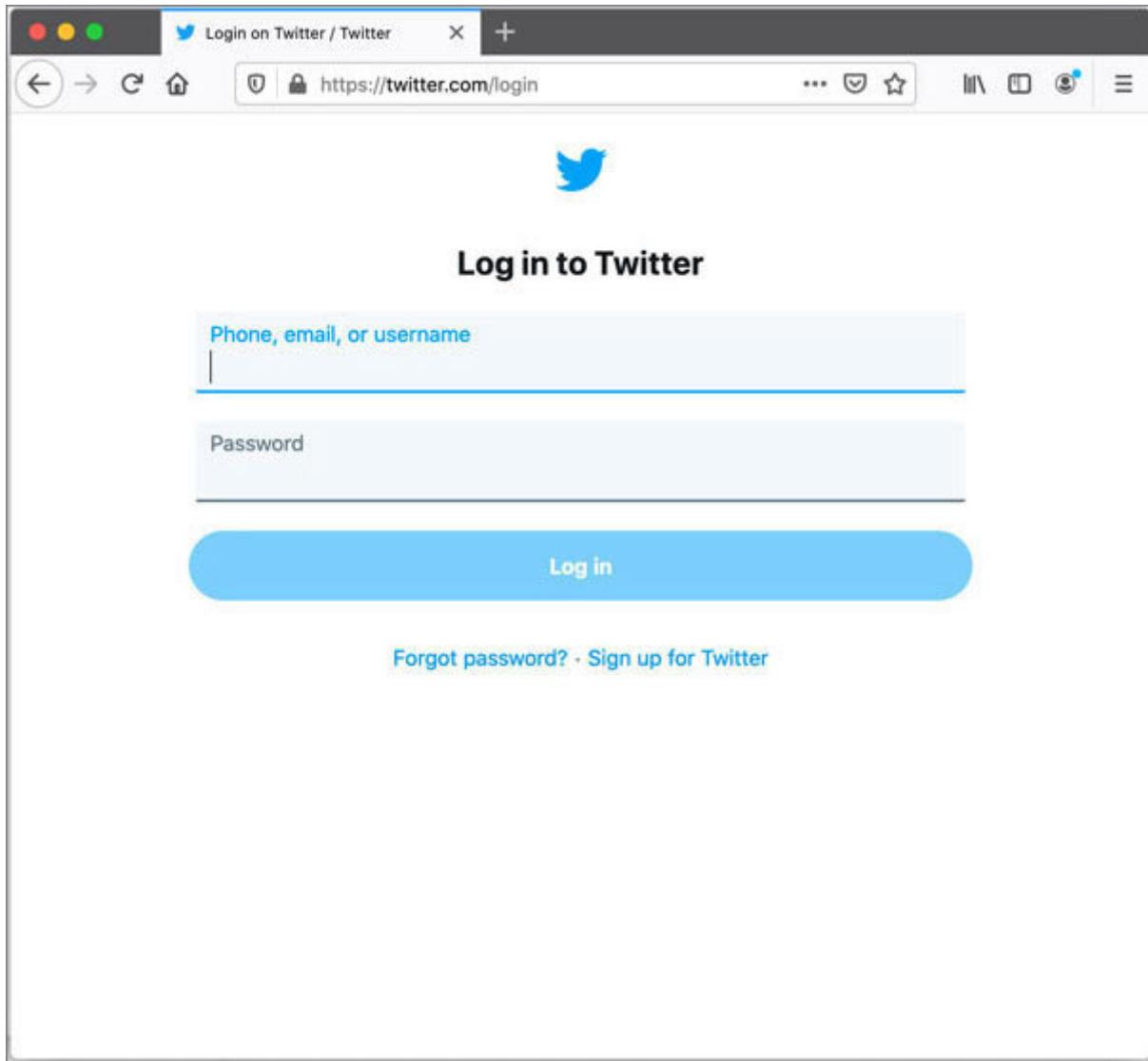


**Table 10.1:** Home Page of <https://developer.twitter.com>

The screen shots above may appear different if different Browsers are used. Also, they may appear different when different machines are used. The appearance in this book is shown from an Apple Mac. The appearance can vary very much on Windows and Linux machines.

To be able to create a Twitter Developer Account, one needs having a Twitter Account. One can create a Twitter account by visiting The process is straight forward and I will not discuss the same. So, the assumption is that you have a Twitter Account.

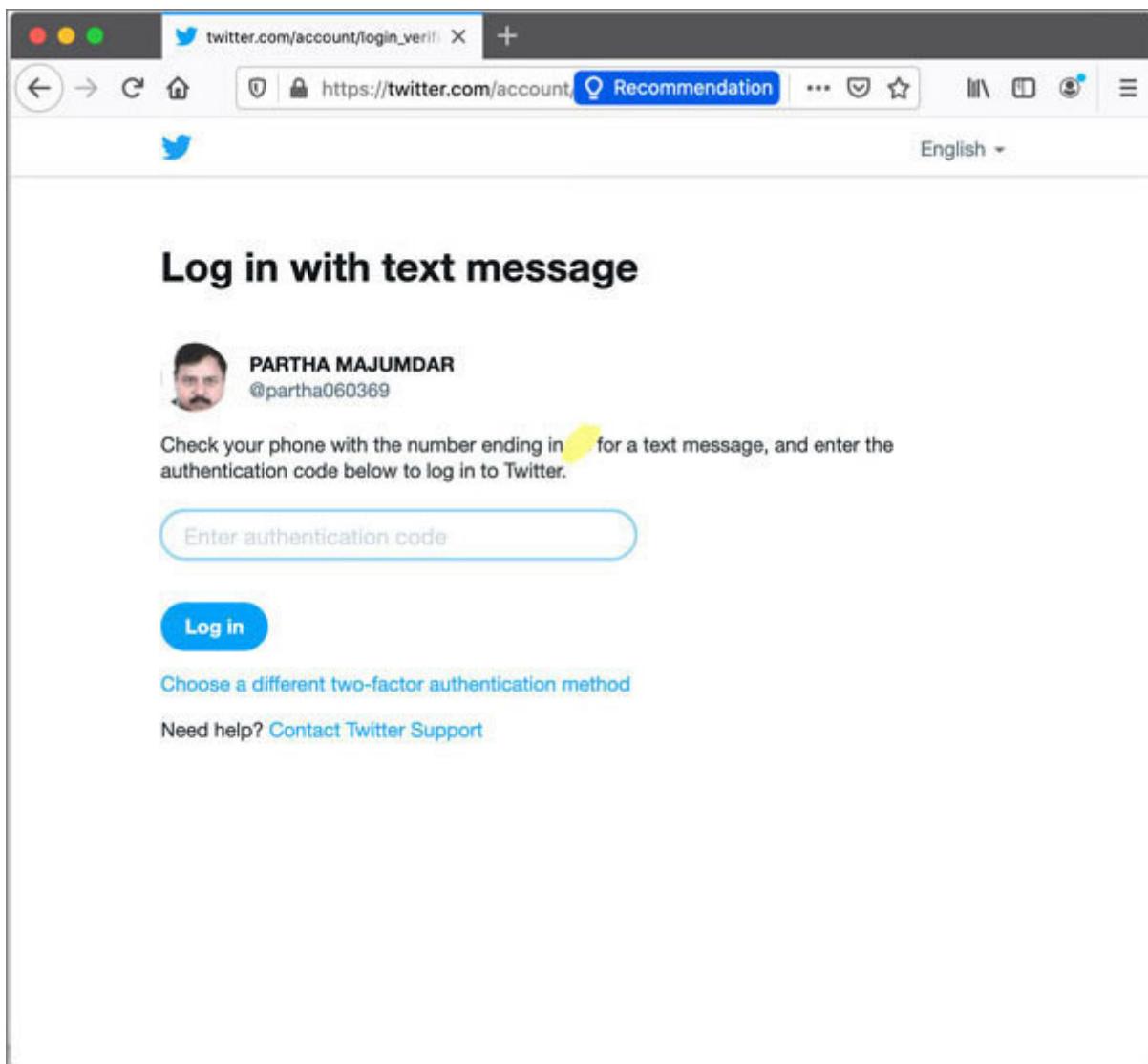
Once you are on the Twitter Developer Home Page as shown in [figure](#) look for the Sign in button as shown in [figure](#). On clicking the Sign in button, the Twitter Login Page will appear as shown in the following screenshot:



**Figure 10.5:** Twitter Login Page

Login to Twitter in the manner you would normally use to login to Twitter by either providing your mobile phone number or an email address or a Twitter username and password.

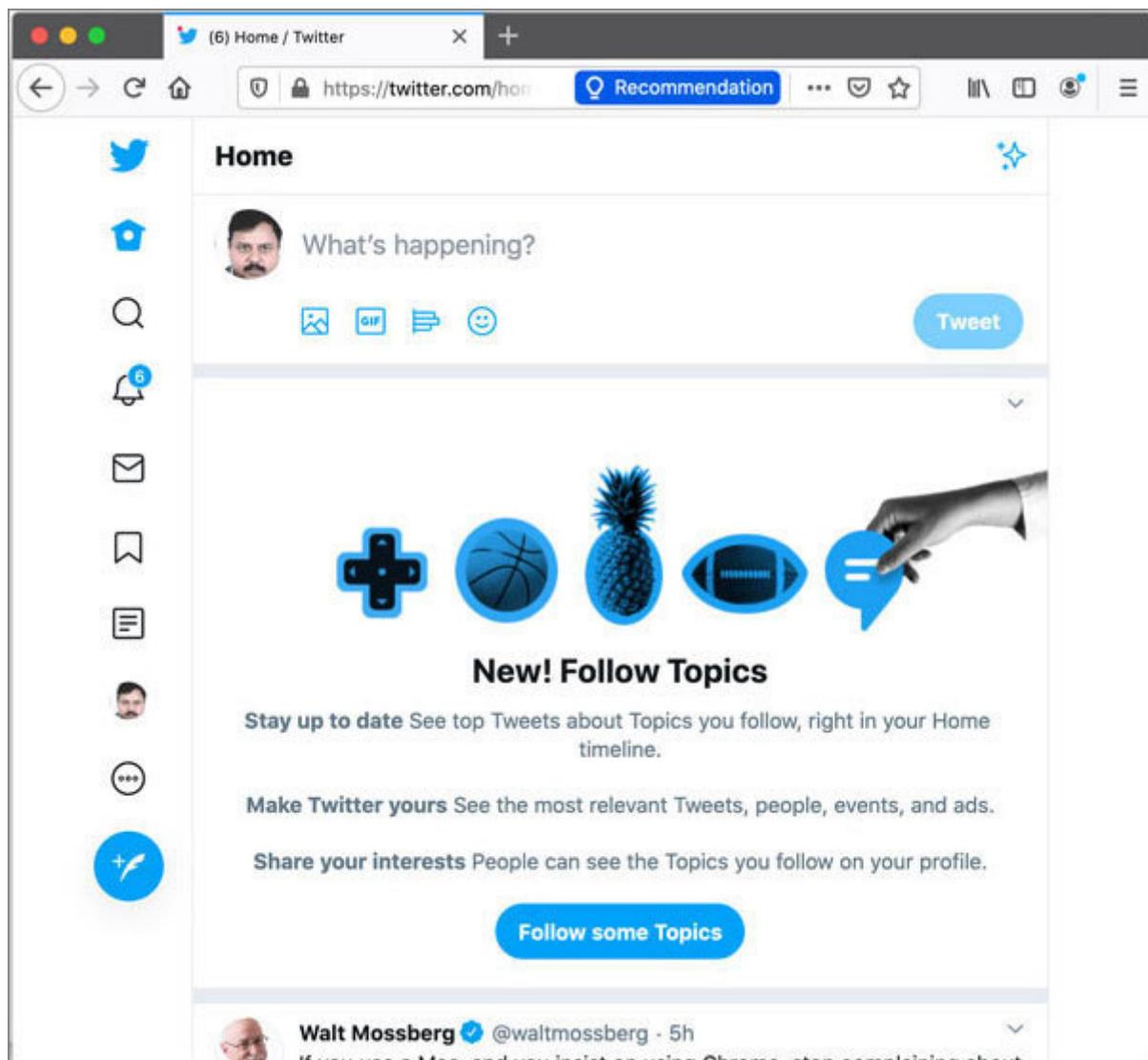
In case you have activated the 2-factor authentication on your Twitter account, Twitter will ask for the authentication code shown as follows. Twitter sends the authentication code to the mobile number associated with the Twitter account:



**Figure 10.6:** 2-factor authentication step for logging into Twitter account

On providing the correct authentication code, Twitter will log into the user account and provide a page like the following screen.

This page is the home page of my Twitter Account:

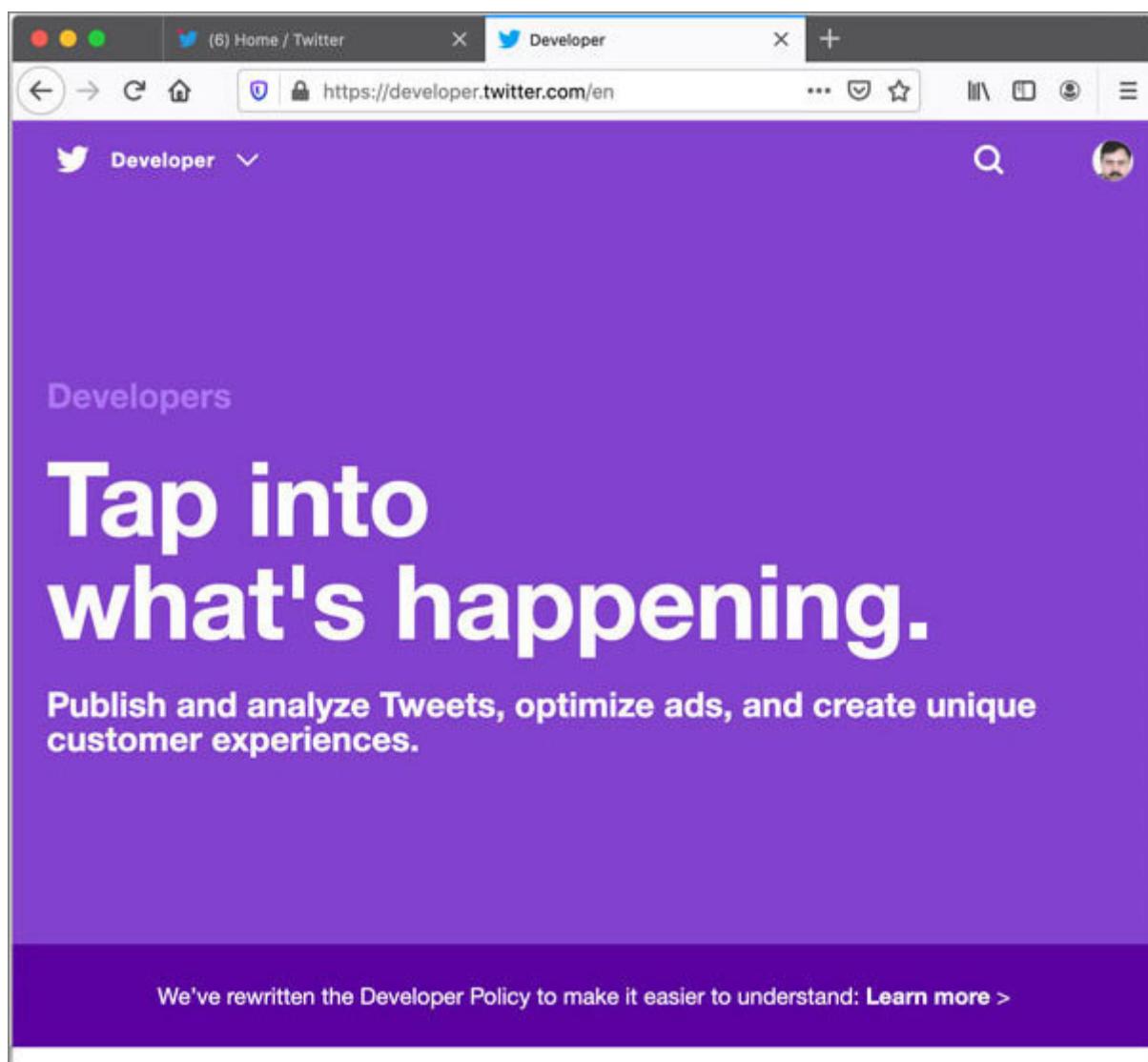


**Figure 10.7:** Twitter Home Page for a User Account

However, we want to access the Twitter Developer Page. For this, we need to once again invoke the URL <https://developer.twitter.com> after having logged into Twitter. So, on a new tab or a new window tab of the browser, type this URL and press

You need to use the same Browser on the same machine which was used to log in to the Twitter Account for invoking the Twitter Developer Home Page once again. The Log In credentials is not carried across different Browsers.

On invoking the Twitter Developer once again, it would appear as follows:



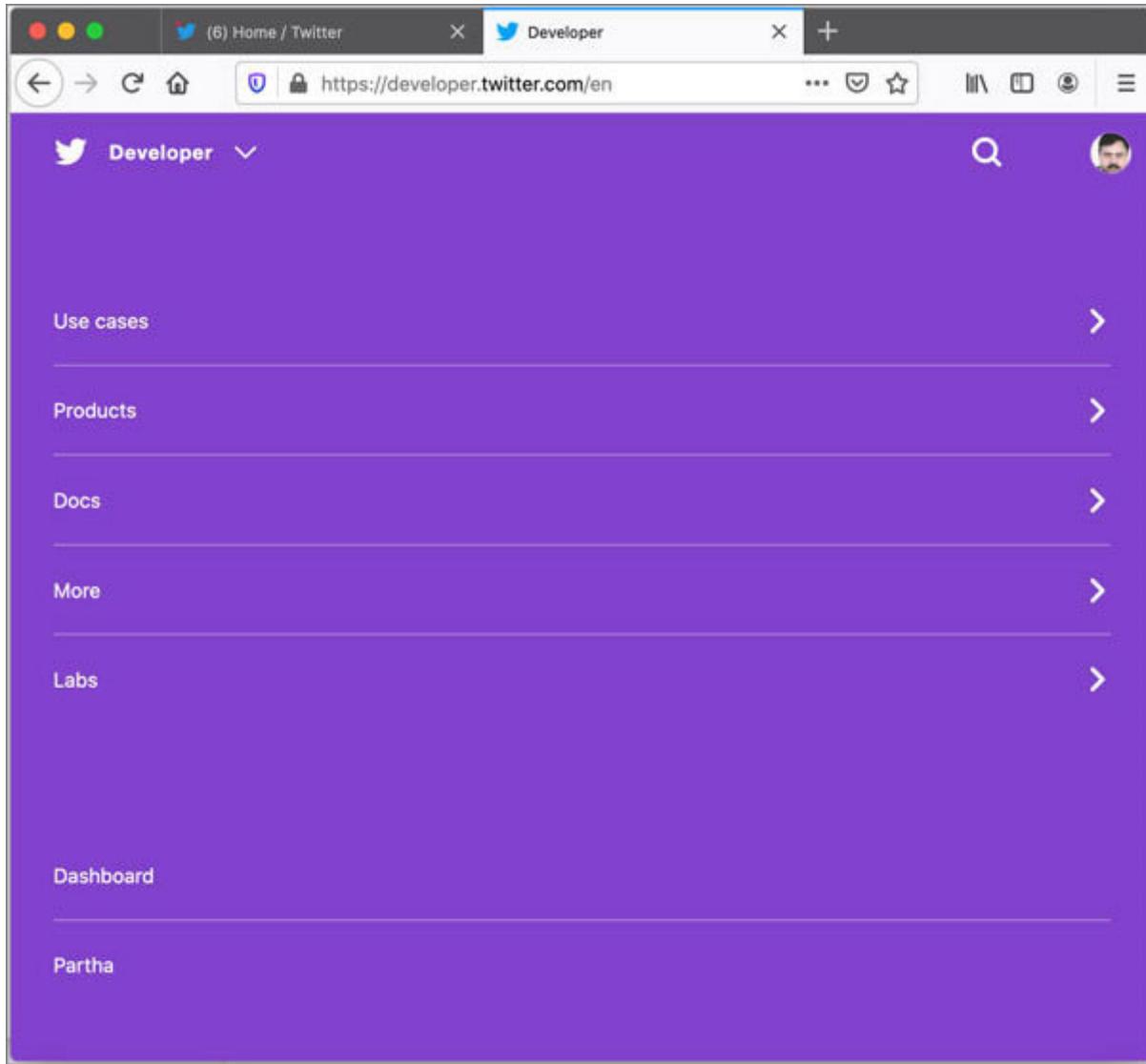
**Figure 10.8:** Twitter Developer Home Page with a Logged in Account

Notice that the logged in user's profile picture' is now displayed on the right-hand top corner of the page.

## [Creating a Twitter Developer Account](#)

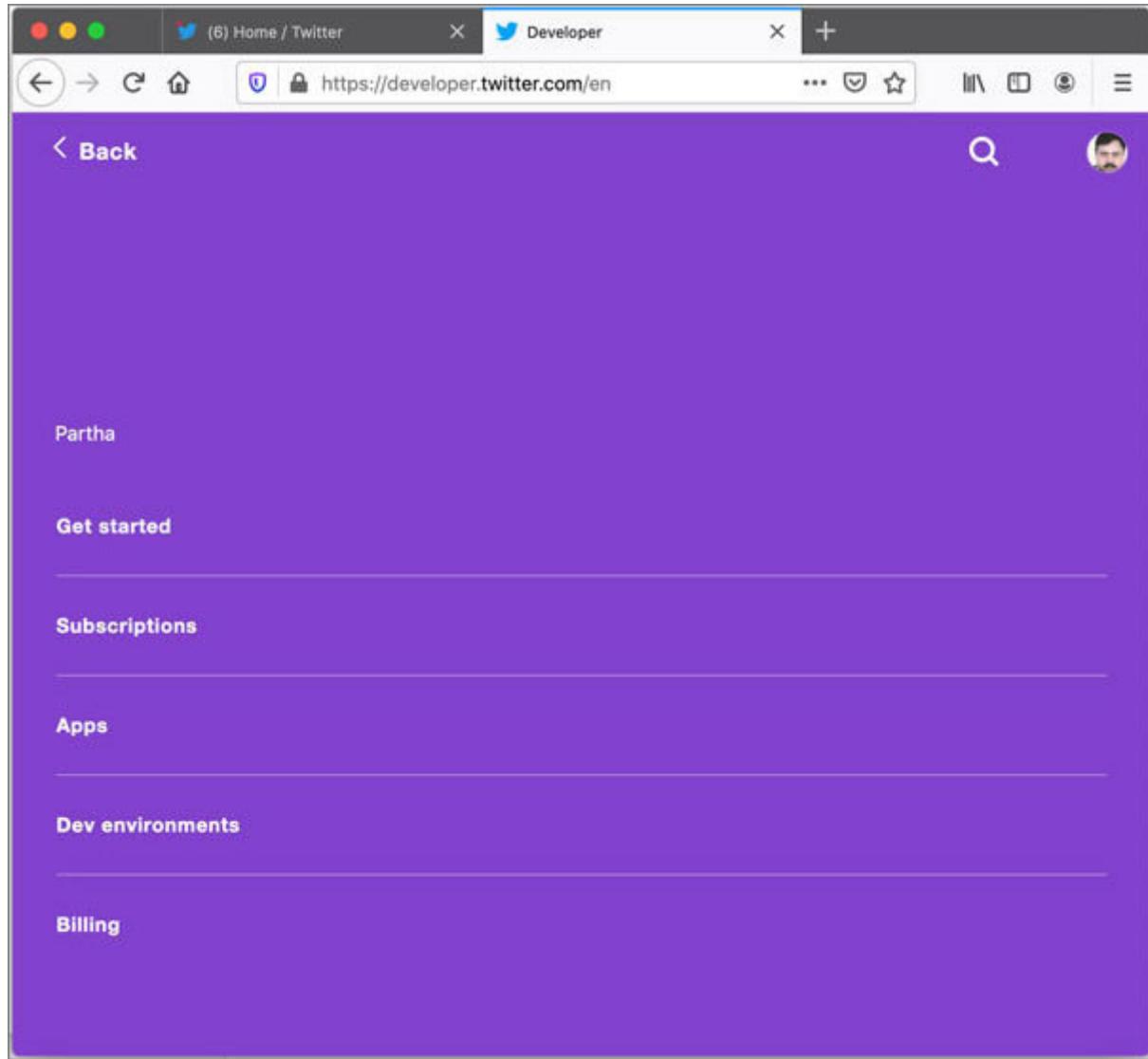
Having logged into the Twitter Developer Home Page, we can create the Twitter Developer Account. We will go through the steps for achieving the same. Notice that in the figure on the top-left corner, there is a drop-down menu under the title This is our point of interest.

Click on this arrow besides the word Developer to get the following menu:



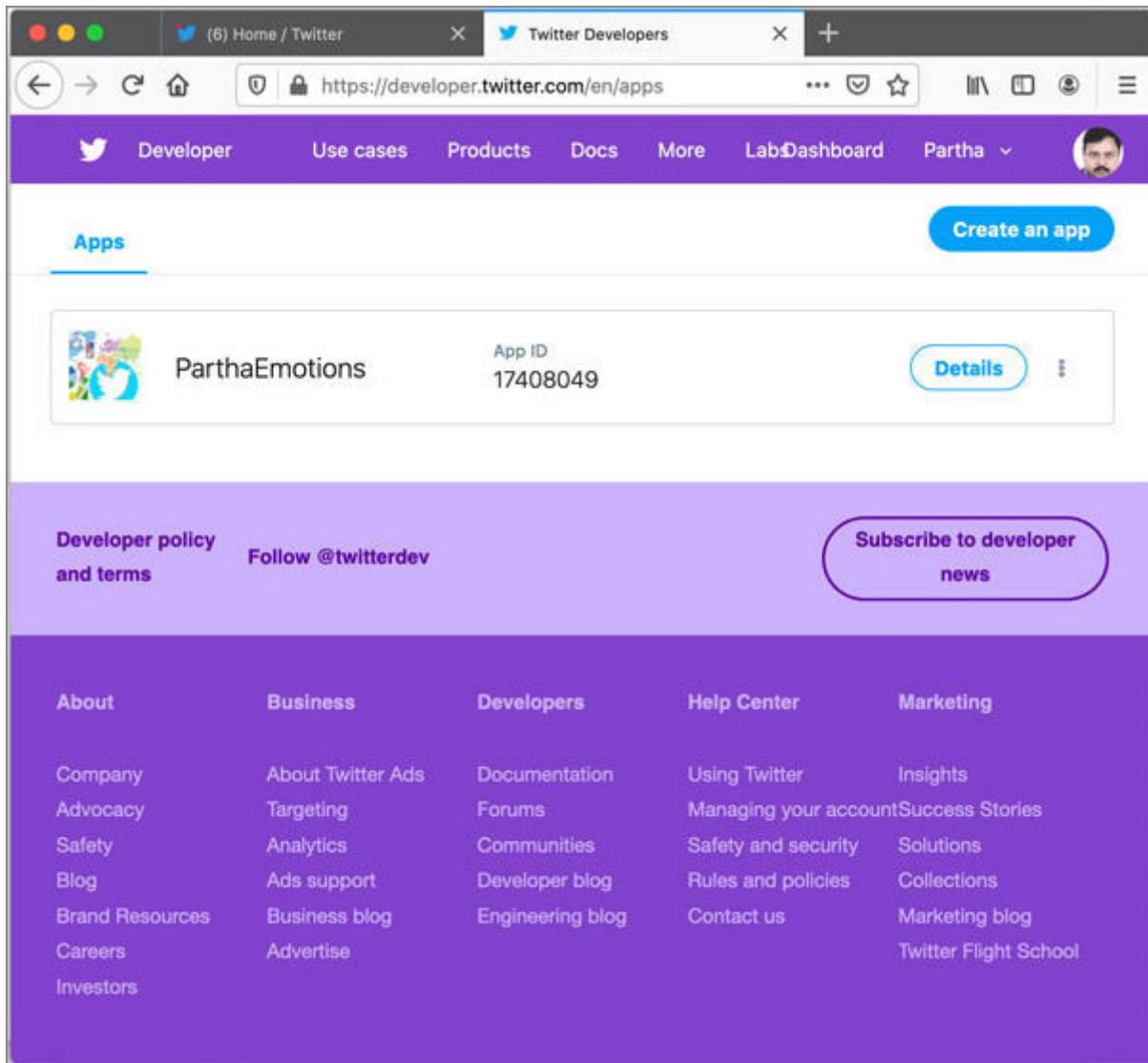
**Figure 10.9:** Menu Items under Developer

Look for the last menu option. It is displayed as **Partha** on the screen. This would be the first name of the Twitter account holder who is logged in. Under this menu option, Twitter allows to make all specific settings for an account, like etc. We would get the following menu on clicking this menu option (in this case, we would click on



**Figure 10.10:** Menu options for Personalizing Twitter Account

On this menu, you can see an option called We need this menu option to create our Twitter Developer Account. We would get the following page on clicking the **Apps** menu page:



**Figure 10.11:** Page on Twitter Developer Setting to start creating a Twitter Developer Account

In the preceding display, you can see that I have created one app called We will start creating a Twitter Developer Account from this page. On the right-hand side, notice that there is a **Create an app** button. We will begin the steps for creating a Twitter Developer Account by clicking this button.

## Steps for creating an app

Now, we will go through the steps for creating an app. This is an essential step. What we are essentially doing is creating a Client App which can interface with Twitter to draw data from Twitter.

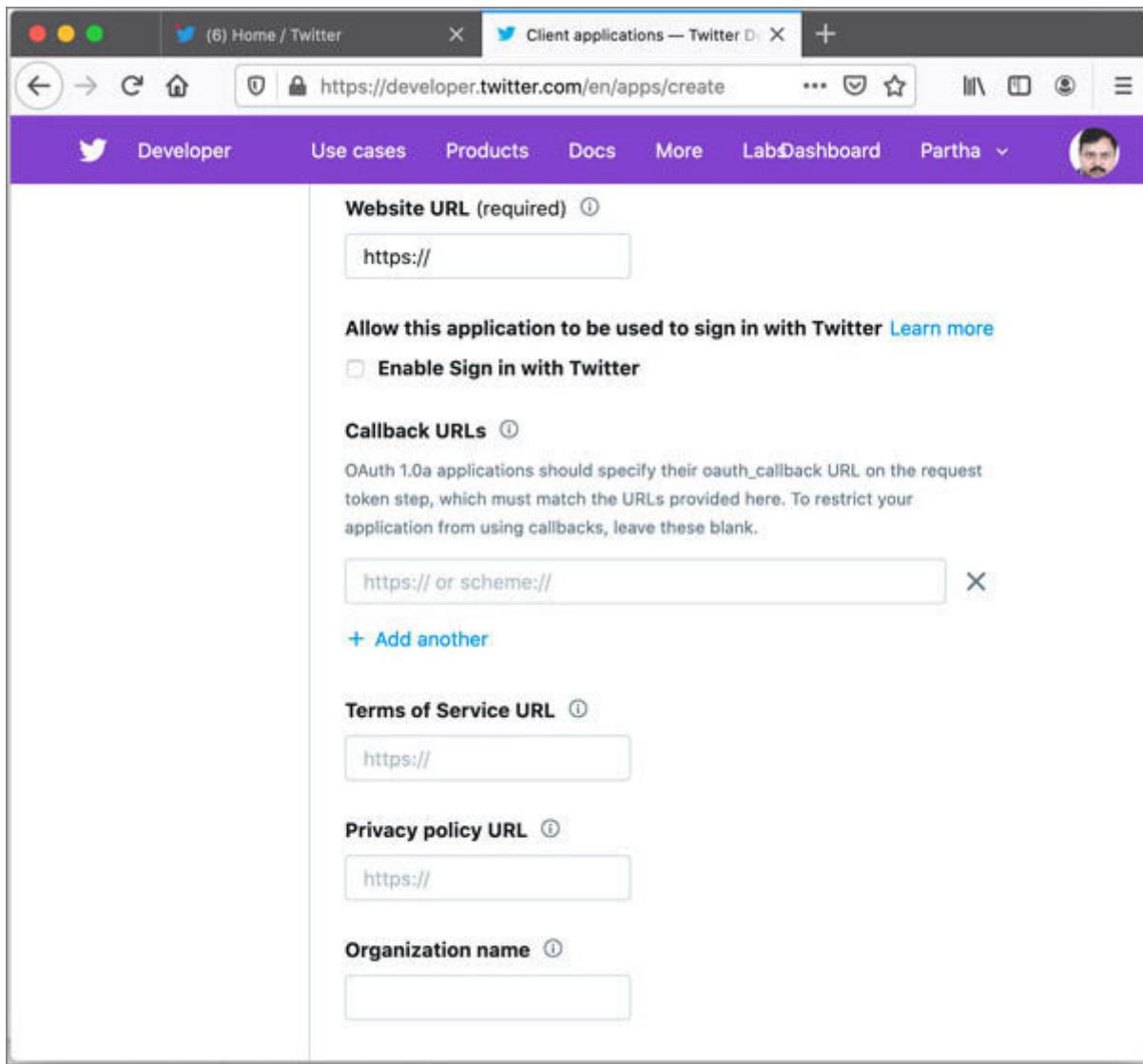
Start by clicking on **Create an app** button. The following page is displayed:

The screenshot shows a web browser window with the Twitter Developer website open at <https://developer.twitter.com/en/apps/create>. The page title is "Client applications — Twitter". The navigation bar includes links for "Developer", "Use cases", "Products", "Docs", "More", "Lab", "Dashboard", and a user profile for "Partha". The main content area is titled "Create an app". On the left, there is a sidebar with links: "Understanding apps", "What is an app?", "Why register an app?", and "Which products require an API key?". The main form starts with "App details" instructions: "The following app details will be visible to app users and are required to generate the API keys needed to authenticate Twitter developer products.". It has fields for "App name (required)" (with a maximum of 32 characters) and "Application description (required)" (with a note: "Share a description of your app. This description will be visible to users so this is a good place to tell them what your app does."). A placeholder text in the description field says "Please be detailed."

***Figure 10.12: Create an app - Screenshot 1***

After this, we need to provide the app details. The first requirement is to provide an app name, which is mandatory. Next, we need to provide the app description, which is also mandatory. The app description needs to be detailed. Twitter assesses the app description very carefully while evaluating whether the app should be permitted on the platform or not. If the app description is found to unsuitable by Twitter, the app will be rejected by them. In this circumstance, the user will not be able to get his/her Twitter Developer Account.

We will see the following as we scroll down this page:



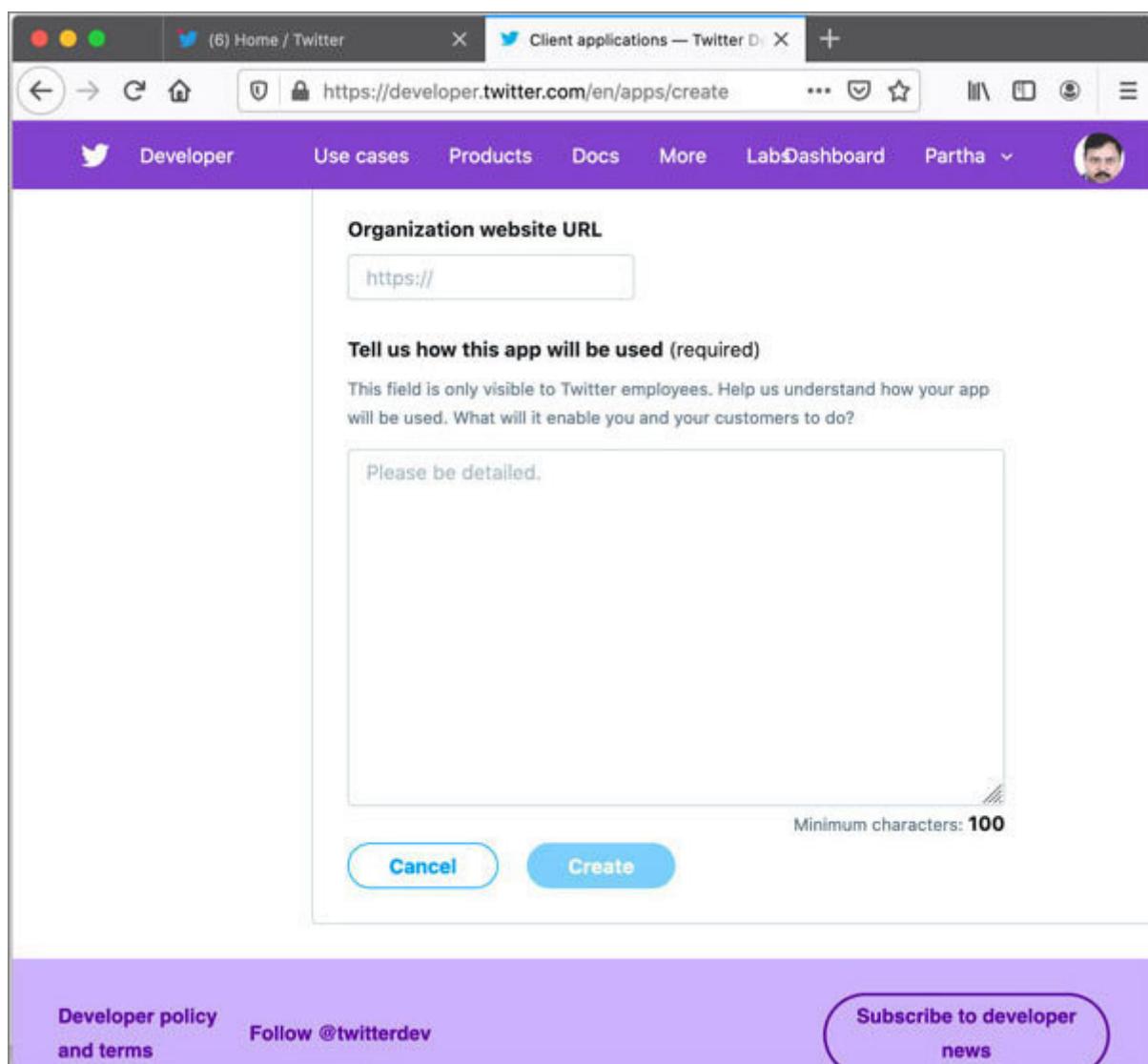
**Figure 10.13:** Create an app - Screenshot 2

Next, we need to provide the **Website**. This should normally be the URL of the website where we will have an application making use of the Twitter data. This is mandatory. We may leave the next checkbox for **Enable Sign in with Twitter** unchecked.

For the next field, **Callback** enter the value **http://127.0.0.1/1410** exactly as it is provided here. We could provide the URLs for **Terms & Conditions** and Privacy Policy at the fields **Terms of**

**Service URL** and **Privacy policy URL** respectively. The simplest way could be uploading these documents to a Shared Drive like Google Drive and providing the associated URLs. It is not mandatory to provide this information.

Next, we could provide the organization name, which is not mandatory. We see the following screen when we scroll down further:



**Figure 10.14:** Create an app - Screenshot 3

In Figure we can see the last two fields on this page that needs to be filled to proceed further. Against **Organization website** we can provide the URL of the organization. However, this is not a mandatory field. One does not need to have his/her own organization to be able to make a Twitter Developer Account.

Now comes the last field - Tell us how this app will be used. This is a mandatory and important field. Here, one needs to provide as much details about the app as possible within 100 characters. The entry in this field is evaluated by Twitter very minutely. The entry in this field mainly decides whether Twitter will allow this app or not. Twitter may send multiple mails to the applicant asking for clarifications for the details provided in this field. Only when Twitter is convinced about how this app will be used, will Twitter allow this App on their platform.

**It is very important to fill each field very carefully. All the information provided is scrutinized very carefully by Twitter before approving the account.**

**As a first step, it is very important to ensure the correctness of all the information.**

**Secondly, all descriptions need to be written very carefully and revised several times. Use suitable key words to make an emphatic impact on the Readers and Evaluators.**

**Lastly, and perhaps most importantly, the intent needs being honest. If any of the provided information is found to be incorrect or abused subsequently, there could be severe consequences.**

Once all the fields have been filled appropriately, click on the Create button. On clicking the **Create** button, the app is submitted to Twitter for evaluation. Twitter normally takes between 3-5 days to complete the evaluation process. The evaluation process can result in either the app being permitted on the Twitter platform or app rejected.

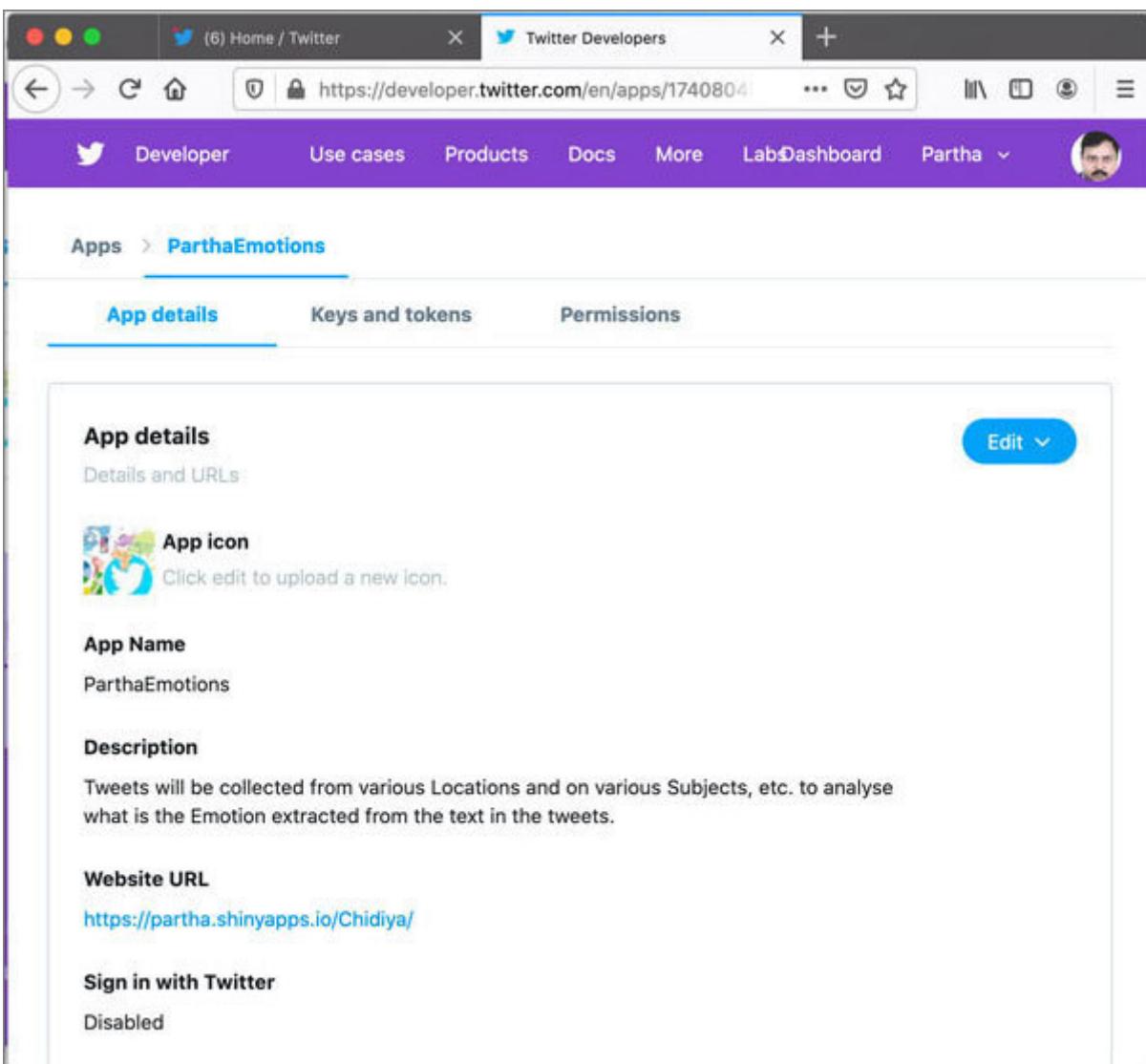
In these 3-5 days, Twitter may send multiple emails asking for various clarifications regarding the information provided. It is important to answer all the questions from Twitter appropriately. It is better to take time to answer these queries. I need to emphasize on being honest in the whole process. Even if Twitter evaluators are unable to detect a dishonest answer during the evaluation process, they constantly keep checking all the aspects of the app. If they find any deviations from whatever the information provided while getting the app approved, they take actions including deactivating the app and taking legal measures. Depending on the nature of the deviation, there could be actions from government agencies as well.

**I cannot comment on whether Twitter has any other considerations other than the details provided this page for approving the app (and thus the Twitter Developer account).**

**Once one has a Twitter Developer Account, he/she could create multiple apps for different purposes.**

## App approval by Twitter

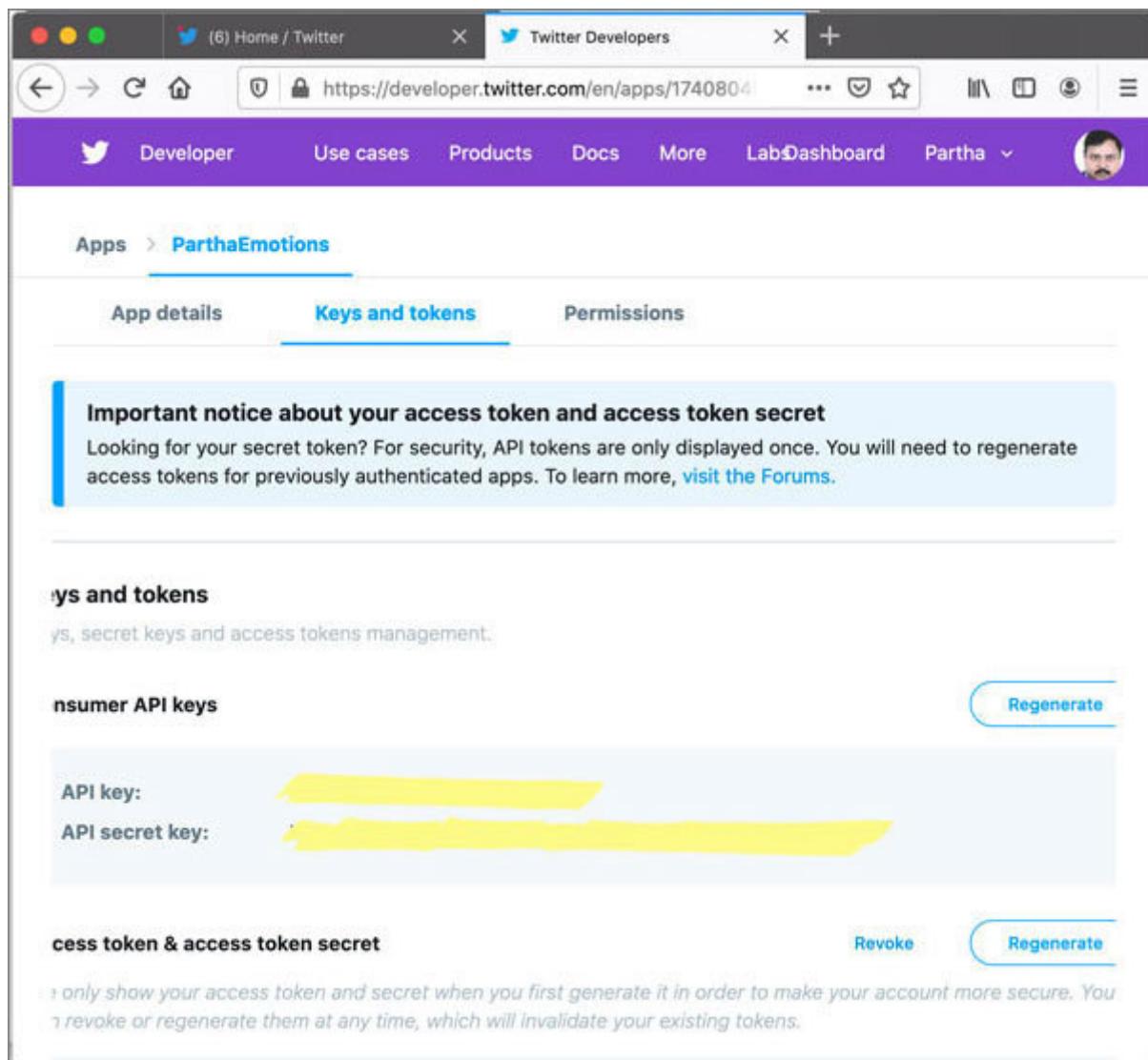
Once the app is approved by Twitter, the details of the app are shown as follows:



The screenshot shows a web browser window with two tabs open: '(6) Home / Twitter' and 'Twitter Developers'. The main content area is the 'Twitter Developers' page, specifically the 'App details' section for an app named 'ParthaEmotions'. The page includes fields for 'App icon', 'App Name' (set to 'ParthaEmotions'), 'Description' (a brief explanation about collecting tweets for emotion analysis), 'Website URL' (a link to <https://partha.shinyapps.io/Chidiya/>), and 'Sign in with Twitter' (set to 'Disabled'). A blue 'Edit' button is visible in the top right corner of the form.

**Figure 10.15:** Approved App on Twitter Developer Account

All the details that you have entered will appear in the **App details** tab. The important thing here is available in the tab **Keys** and When you click on this tab, you would see something as shown in the following screenshot:



**Figure 10.16:** Keys and tokens tab for the created App – Screenshot 1

There are **four** important information on this page that we need for creating our program in R (or any other programming language). The first two are shown in [figure](#). They are API key and

API secret key. I have masked these values as they are private for me. I would advise you to also keep these values confidential and not share with anyone. It is important to copy these values and store them safely for creating any application for accessing Twitter data.

When we scroll down this page, we would see something as shown in the following screenshot:

The screenshot shows a web browser window with the Twitter Developers website open. The URL in the address bar is <https://developer.twitter.com/en/apps/1740804>. The page displays the 'Keys & tokens' section for a specific app. It shows the following information:

| Key                  | Value                                  | Details                     |
|----------------------|--|-----------------------------|
| Access token:        | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX         | Last generated: Feb 8, 2020 |
| Access token secret: | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |                             |
| Access level:        | Read and write                         |                             |

Below this, there are links for 'Developer policy and terms', 'Follow @twitterdev', and 'Subscribe to developer news'. At the bottom, there is a navigation menu with links like 'About', 'Business', 'Developers', 'Help Center', and 'Marketing', each with further sub-links.

**Figure 10.17:** Keys and tokens tab for the created App – Screenshot 2

Here we have two more important information and they are **Access token** and **Access token**. It is advised to not share this information with anyone. Also, it is advised to store these values securely.

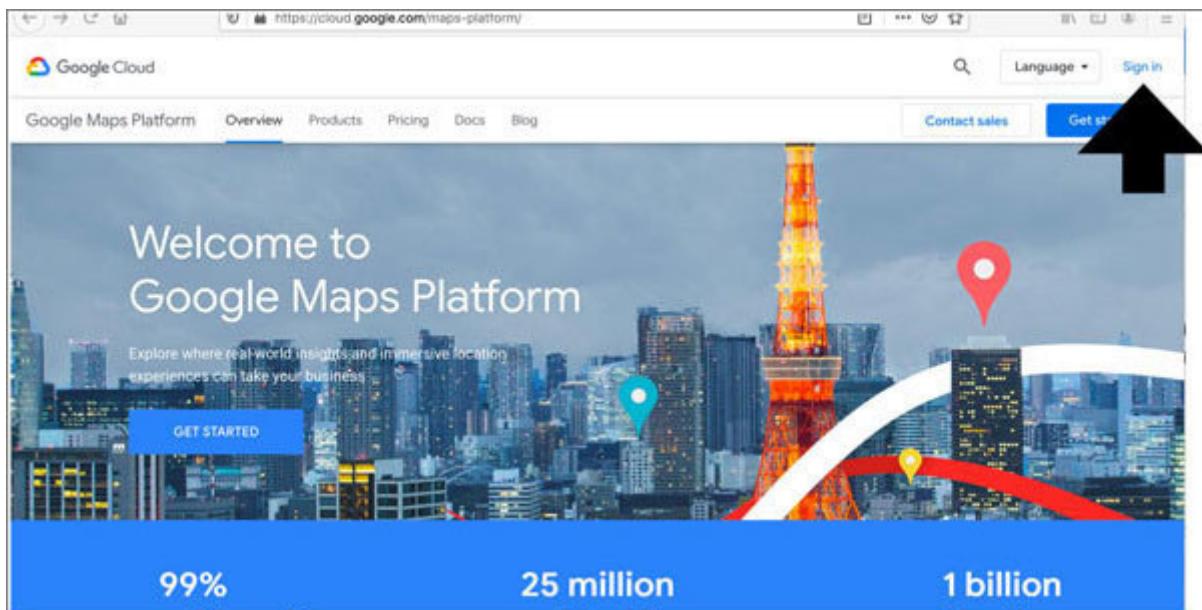
We will see in [Chapter 11: Emotion Analysis on Twitter Data](#) how to use these four values while developing our application using R programming.

## ***Google Maps API key***

Twitter data contains the location from where a Tweet was made. To be able to extract the Twitter data for a location, we need Google Maps API Now we will go through the steps required for obtaining a Google Maps API key.

## URL for creating a Google Maps API key

Visit the URL <https://cloud.google.com/maps-platform> to create a Google Maps API key. On invoking this URL from Google Chrome, you should get a UI like the one shown in the following screenshot:



**Figure 10.18:** Google Maps Platform Home Page

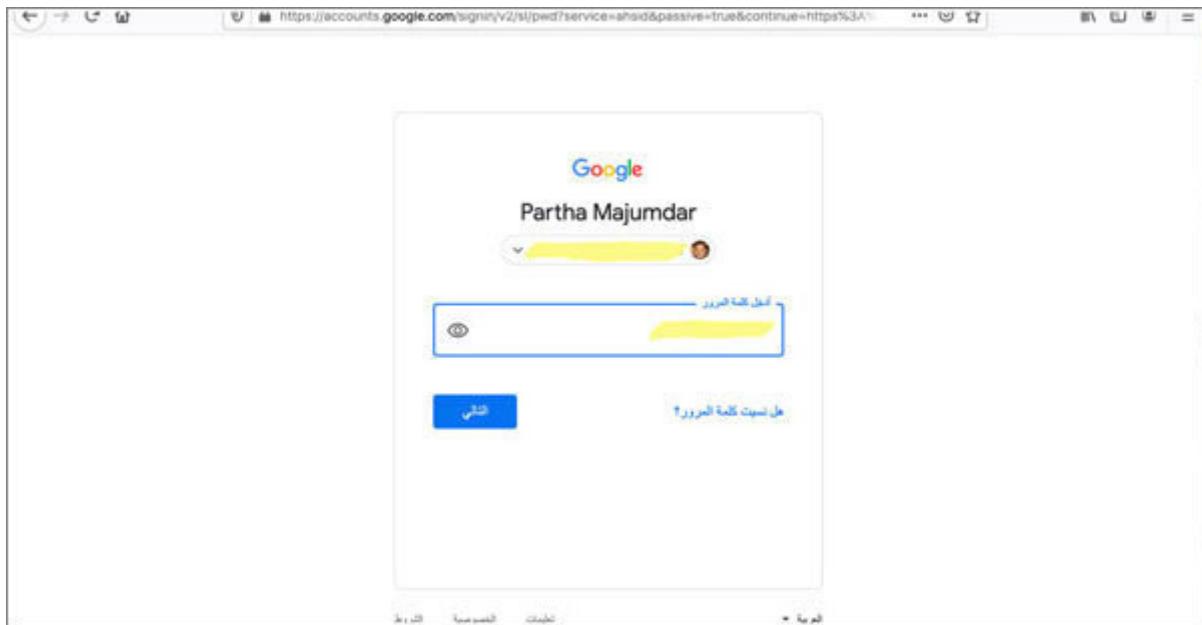
On this page, click on the **Sign In** button as pointed by the black arrow in [figure](#). You will be redirected to the **Google Sign In** page shown as follows:



**Figure 10.19:** Google Sign In Page

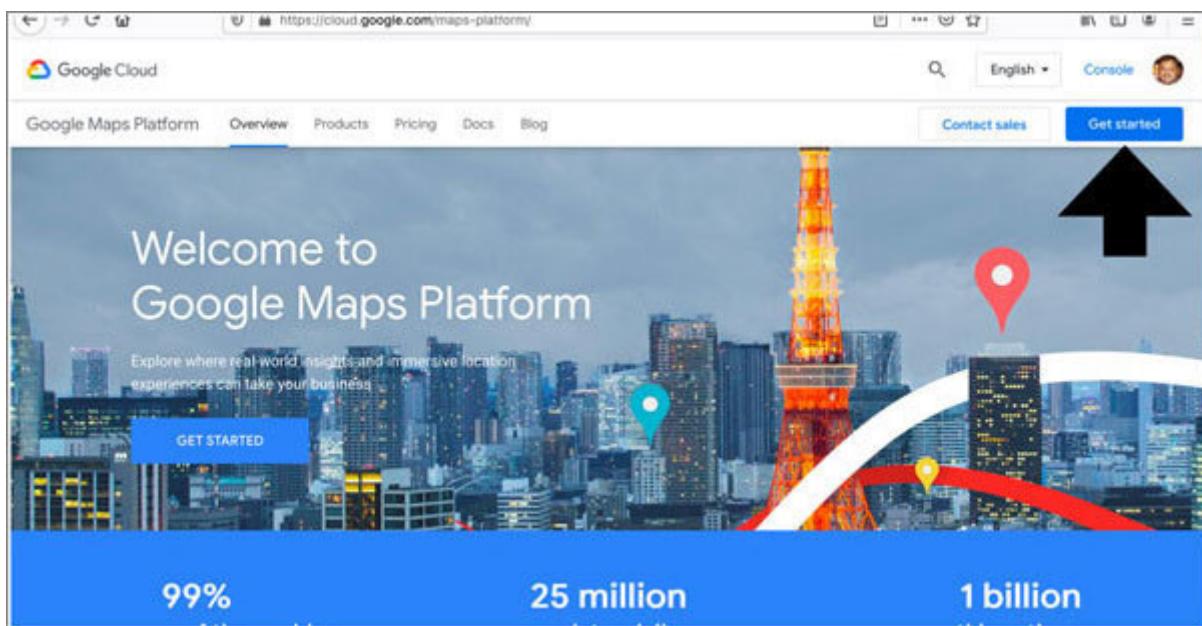
The display in [figure 10.19](#) will differ based on the location from where you are signing in. I am writing this book from Riyadh and thus you see the Arabic text.

On selecting your account, you would need to provide the password to log into the account:



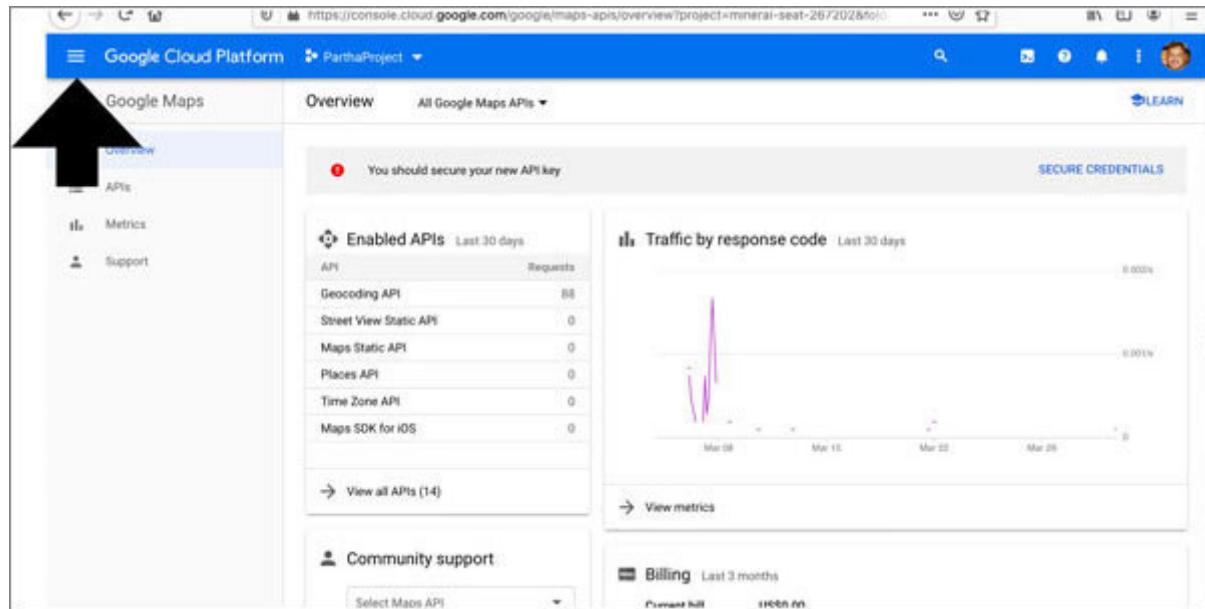
**Figure 10.20:** Google Sign In password entry page

On logging into the Google account, you should see the following page:



**Figure 10.21:** Google Map platform home page after logging in with Google Account

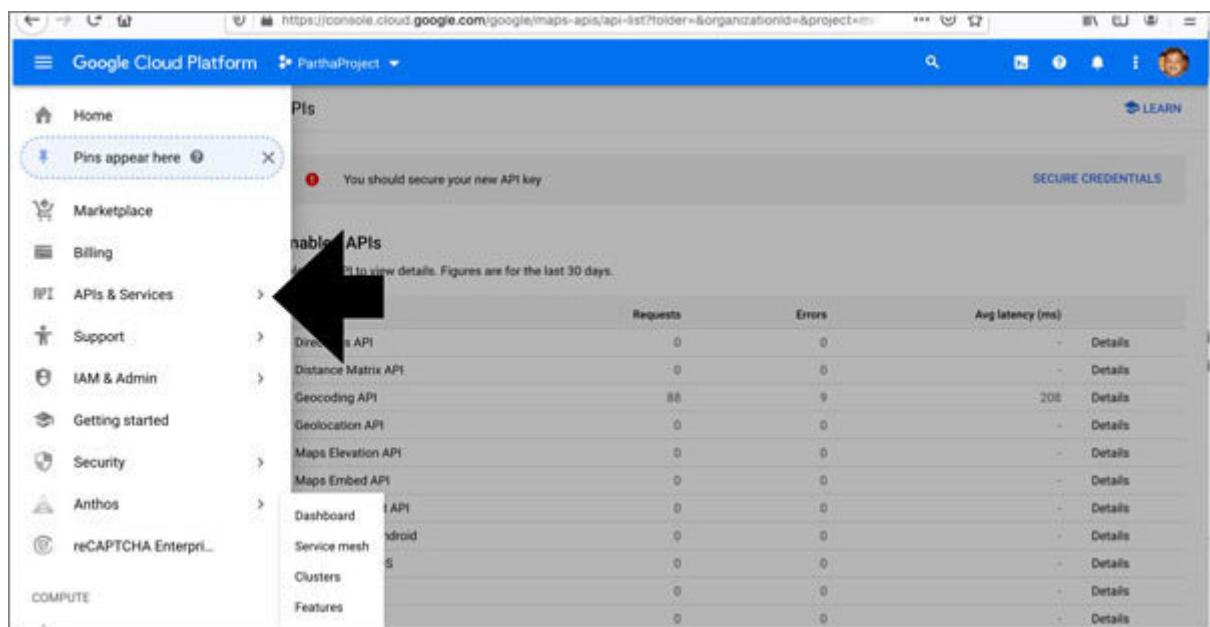
Click on the **Get Started** button as pointed by the black arrow in Figure. You should see the page like the one shown as follows:



The screenshot shows the Google Cloud Platform Google Maps API dashboard. At the top, there's a blue header bar with the URL <https://console.cloud.google.com/google/maps-apis/overview?project=mineral-seat-267202&tab=apis>. Below the header, the title "Google Cloud Platform" and the project name "ParthaProject" are visible. On the left, a sidebar menu includes "Google Maps" (selected), "Overview", "Metrics", and "Support". The main content area has three sections: "Enabled APIs" (last 30 days) showing Geocoding API (88 requests), Street View Static API (0), Maps Static API (0), Places API (0), Time Zone API (0), and Maps SDK for iOS (0); "Traffic by response code" (last 30 days) with a chart showing a single purple spike on March 8th; and "Billing" (last 3 months) with a "Current Bill" of \$0.00 and a "Next Bill" of \$0.00. A "SECURE CREDENTIALS" link is in the top right.

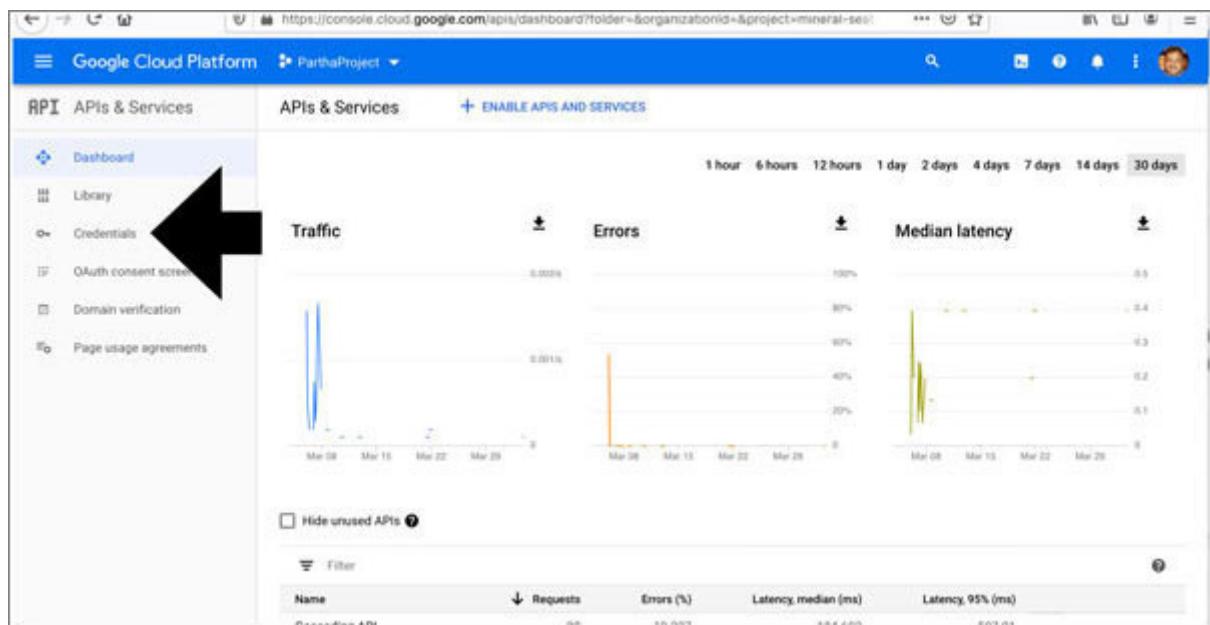
**Figure 10.22:** Google Maps Dashboard Page

Invoke the menu as pointed by the black arrow in figure. The menu will look like the following screenshot:



**Figure 10.23:** Google Maps menu

On this menu, select the option for **APIs & Services** as pointed by the black arrow in [figure](#). You will get the following menu:



**Figure 10.24:** Google Maps APIs and services menu

Click on the **Credentials** menu option as pointed by the black arrow in [figure 10.24](#) to get the following page:

The screenshot shows the Google Cloud Platform Credentials page. On the left, there's a sidebar with options like Dashboard, Library, Credentials (which is selected and highlighted in blue), OAuth consent screen, Domain verification, and Page usage agreements. The main area has a header with 'Credentials' and a '+ CREATE CREDENTIALS' button. Below it, there's a section for creating credentials with a 'Learn more' link and a note about configuring a consent screen. A red circle highlights the 'API keys' section, which lists three entries: 'API key 3' (created 2 Apr 2020, restrictions: None, usage: 0), 'Maps API Key' (created 6 Mar 2020, restrictions: None, usage: 79), and 'API Key' (created 4 Feb 2020, restrictions: None, usage: 9). There's also a section for 'OAuth 2.0 Client IDs' which is currently empty.

**Figure 10.25:** Google Maps credentials page

This page lists all the Google Maps API keys for the account. You can create a new Google Maps API key by clicking on **CREATE CREDENTIALS** button as pointed by the black arrow in [figure](#). You can copy the Google Maps API key. Do not share the Google Maps API key with anyone. Store the Google Maps API key securely.

**Usage of Google Maps using the Google Maps API Key is a PAID service.**

## Conclusion

In this chapter, we discussed about the enormity of data available from Twitter. We discussed a potential application that can be built using the Twitter data. For being able to analyze Twitter data, we need to have a Twitter Developer Account. Also, it is useful to have a Google Maps API key. We discussed how to create a Twitter Developer Account. We also learned how to obtain a Google Maps API Key.

In the next chapter, we will find out more about R code that can be used to fetch Twitter data and analyze data obtained from Twitter.

### Points to remember

More than 500 million Tweets are posted daily.

The 4 information that we need for programming for analyzing Tweets are:

- API Key
- API secret key
- API token
- API token secret

Do not share any of the keys and store them securely.

### Multiple choice questions

The Twitter Mobile App was launched on:

2006

2010

2006

2010

What is the length of the Twitter App API key?

21 characters

25 characters

15 characters

18 characters

What is the length of the Google Maps API key?

21 characters

28 characters

15 characters

39 characters

## Answers to MCQs

C

B

D

## Questions

Write one application you would like to create that will use data from Twitter.

### Key terms

**API:** Application Programming Interface is a software intermediary which allows two applications to talk to each other.

## CHAPTER 11

### Emotion Analysis on Twitter Data

In [Chapter 10: Introduction to Twitter Data Analysis](#), we discussed the pre-requisites to analyze the data obtained from Twitter. In this chapter, we will go through the details on how to conduct analysis on the data obtained from Twitter. Before that, we will discuss the data that we can obtain from Twitter and the ways to do so.

We will see that there is an enormous amount of data that we can obtain from Twitter on many attributes. As a result, there is an enormous amount of analysis possible on this data. However, we will not discuss all kinds of analyses possible to be conducted on this data as that would be a gigantic task. So, the effort will be to introduce the data and discuss some key programming techniques for analyzing it. Using the knowledge of the data and the programming techniques, you should be able to perform almost any kind of analysis on the data obtained from Twitter or on the data obtained from any other source.

Our goal is to be able to extract emotion expressed in tweets. So, progressively, we will move towards this goal in this chapter.

## Structure

In this chapter, we will discuss the following topics:

A few features of Twitter

Fetching data from Twitter

- Library required for fetching tweets
- Setting up the Twitter account
- Authenticating the Twitter account
- Fetching tweets for search string(s)
- Fetching tweets from a location

Displaying tweets using DT

Analyzing data obtained from Twitter

- Time Series Charts for when tweets were posted
- Country analysis

- Place analysis
- Language analysis
- User analysis
- Source analysis
- Hashtag analysis
- Location analysis with maps

Emotion analysis of Tweets

## Objectives

After studying this unit, you should be able to:

Fetch data from Twitter

Understand the data returned by Twitter

Conduct analysis on the data returned by Twitter

Extract emotions from a set of tweets

## A Few Features of Twitter

Twitter provides several features to its users. This discussion should help understand the data returned by Twitter. Twitter provides a feature that a reader of a Tweet (published tweet (by anyone) can use to reply against the The same user can post multiple replies against a tweet. Further, any number of readers of the tweet can post a reply. Readers of the tweet can also post replies against the replies.

Twitter provides a feature to its users to be able to repost a tweet posted by another user. Reposting a tweet of another User is called A Quote Tweet is a retweet that has been made with a comment. It is different to a regular retweet because it will be shown to your followers along with your comments. It starts a new thread that people can retweet and like, separate to the original tweet.

**Favorites** are described as indicators that a tweet is well-liked or popular among online users. A tweet can be identified as a favorite by the small star icon ( ) seen beside the post. Twitter users can mark a tweet as a favorite to let the author know that they like it.

People use the Hash Tag symbol (#) before a relevant keyword or phrase in their Tweet to categorize those and help them show more easily in Twitter search. The **symbol** allows a Twitter user to

*tag* another Twitter user and notifies them that they've been mentioned.

Tens of millions of links are posted in tweets each day. Wrapping these shared links helps Twitter protect users from malicious content while offering useful insights on engagement. All links submitted within tweets and direct messages, regardless of length, will eventually be wrapped with the t.co shortened.

When one protects his/her tweets, he/she will receive a request when new people want to follow him/her, which he/she can approve or deny. The Protected Tweets, including the permanent links to the Protected Tweets, will only be visible to the followers of the person who tweeted. The Protected Tweets will only be searchable on Twitter by the person who tweeted and by his/her followers.

## Fetching data from Twitter

Now we will start our discussion on how to fetch data from Twitter.

### **Library required for fetching tweets**

There are a few R libraries available for fetching tweets from Twitter. We will discuss the use of the library I have found this library to be reliable and it is updated regularly.

#### **Documentation for rtweet library can be obtained from**

To include the rtweet library in an R program, we need the following line of code:

```
library(rtweet)
```

## Setting up a Twitter account

To be able to fetch tweets from Twitter, we need to set up a Twitter Developer Account in our R program. We can do so using the following code:

```
##### SET UP THE TWITTER ACCOUNT
#####
api_key <- "XXXX"
api_secret_key <- "XXXX"
access_token <- "XXXX"
access_token_secret <- "XXXX"
```

We declared four variables for the four keys as provided by the app we created in [Chapter 10: Introduction to Twitter Data Analysis](#) while creating the Twitter Developer Account. In the preceding code, the XXXX needs to be replaced by the associated keys obtained from the App Keys and Tokens. Notice that all the four key values need to be enclosed in quotes as these are string values.

Writing the preceding code exposes the Twitter app keys to anyone who can read this code. This may not be the best thing to do. You may create multiple R programs using the same set of App Keys and Tokens.

A better and recommended practice is to define these keys and tokens in a text file shown as follows:

Parameter,Value

DBUserID,xxxx

DBPassword,xxxx

api\_key,xxxx

api\_secret\_key,xxxx

access\_token,xxxx

access\_token\_secret,xxxx

dsn\_driver,com.ibm.db2.jcc.DB2Driver

dsn\_database,xxxx

dsn\_hostname,xxxx

dsn\_port,50000

dsn\_protocol,TCPIP

google\_maps\_api\_key,xxxx

I have named this file as You can give any name of your choice to this file. SystemParameters.csv is a **comma separated values** file with two columns of values – the first column is named Parameter and the second column is named So, this file contains a list of key-value pairs. As an example, dsn\_driver is the key and com.ibm.db2.jcc.DB2Driver is the value.

**I have replaced the values of the column Value with In your application, you must replace all the xxxx with the actual values.**

This file must be stored on the server from where the application will run. However, we could encrypt this file so that nobody could read the contents and these contents would be safe. However, for

the moment, we will just keep this file as shown in the preceding code.

To access the Twitter Keys and Tokens from this file, we must first read this file. We can do so using the following code:

```
##### READ THE SYSTEM PARAMETERS
#####
```

```
v_system_parameters <- read.csv(file = "SystemParameters.csv",
header = TRUE)
attach(v_system_parameters)
```

We read the contents of the file into the data frame So, this data frame has two columns – Parameter and Now, we can extract the Twitter Key and Tokens using the following code:

```
##### SET UP THE TWITTER ACCOUNT
#####
api_key <- as.character(v_system_parameters[(which(Parameter ==
"api_key")),]$Value)
api_secret_key <-
as.character(v_system_parameters[(which(Parameter ==
"api_secret_key")),]$Value)
access_token <- as.character(v_system_parameters[(which(Parameter ==
"access_token")),]$Value)
access_token_secret <-
as.character(v_system_parameters[(which(Parameter ==
"access_token_secret")),]$Value)
```

## Authenticating the Twitter account

We can authenticate the Twitter app before fetching data from Twitter by using the Keys and Tokens. For doing so, we can use the `create_token()` function in the `rtweet` library shown as follows:

```
## Authenticate
token <- create_token(
  app = "ParthaEmotions",
  consumer_key = api_key,
  consumer_secret = api_secret_key,
  access_token = access_token,
  access_secret = access_token_secret
)
```

Note that for the parameter we need to provide the app name created using the Twitter Developer Account. In my case, the app name is You need to replace this with the name of your app.

For the remaining four parameters – and we need to provide the and respectively [that we discussed in [Chapter 10: Introduction to Twitter Data](#)

**We can fetch data from Twitter without authenticating. However, there will be restrictions on what data can be fetched and how much data can be fetched.**

We can view the token generated as follows. I have hidden the value of the key:

```
token
##
##
##   request:    https://api.twitter.com/oauth/request_token
##   authorize:  https://api.twitter.com/oauth/authenticate
##   access:     https://api.twitter.com/oauth/access_token
## ParthaEmotions
##   key:        WXXXXXXXXXjVxXXXXXXXXXeD
##   secret:
## oauth_token, oauth_token_secret
## ---
```

## Fetching tweets by search string(s).

We can fetch tweets using the `search_tweets()` function in the `rtweet` library. We will discuss the first form of using this function in the following code where we fetch tweets based on a search string:

```
v_temp <- search_tweets(  
  q = "Coronavirus",  
  '-filter' = "replies",  
  n = 1000,  
  include_rts = FALSE,  
  retryonratelimit = FALSE  
)
```

The parameter `q` is the query string. In the preceding example, the query string is So, the above command will fetch tweets having the word `Coronavirus` mentioned in them.

The query string can be any string. It can be a hashtag. For example, if we write `q = "#COVID"` then the tweets containing the hashtag `#COVID` will be fetched.

The query string can contain more than one word like `Virus = "Corona Virus"`. In this case, the `search_tweets()` function will search for both the words – `Corona` and `Virus` - and fetch tweets containing both the words.

To search for either of the words in a query string, it could be written as q = "Corona OR

To search for exact phrases, the query string needs to be enclosed in quotes like Then, we need to write q = ""Corona Virus"" (enclosing the double quoted phase in a pair of single quotes) or as q = "\"Corona Virus\\"" (Backslash is the escape character).

Lastly, the query string can contain both a string and a hashtag like q = "Corona Virus OR

The parameter '-filter' = "replies" excludes the replies made against the tweets fetched. Notice that you need to use the backquotes around the -filter command. The parameter **n** defines the number of tweets to fetch. So, when we state n = the search\_tweets() function will try to fetch 1,000 Tweets. However, the actual number of Tweets fetched may not be exactly 1,000. It can be more than 1,000 or less than 1,000. The number of tweets fetched depends on how many tweets matching the search criteria are found within the search window.

**search\_tweets()** function will search Twitter from the latest tweet and moves down the tweets based on when the Tweets were posted. Within each Tweet analyzed by search\_tweets() function, it will try to find the query string.

The parameter include\_rts takes a Boolean value of either TRUE or FALSE. When it is set to FALSE, retweets are not fetched; retweets are fetched

otherwise. The parameter `retryonratelimit` takes a Boolean value of either TRUE or FALSE. This parameter is useful when fetching a large number of tweets. When a large number of tweets are fetched, like 50,000 or 3 million, etc., Twitter cannot return so many tweets in one call. Instead, Twitter will return these Tweets in installments if the `retryonratelimit` parameter is set to TRUE. The result set of each installment is added to the same output data frame.

**Note:** Twitter Rate Limits cap the number of search results returned to 18,000 every 15 minutes. The complete documentation for all the options of `search_tweets()` function can be found at [https://www.rdocumentation.org/packages/rtweet/versions/0.7.0/topics/search\\_tweets](https://www.rdocumentation.org/packages/rtweet/versions/0.7.0/topics/search_tweets)

## VIEWING THE FETCHED TWEETS

The `search_tweets()` function returns a data frame containing the tweets. Twitter returns 90 columns of data. We know that we can see the column names of the columns in a data frame by using the `colnames()` function shown as follows:

```
colnames(v_temp)
## [1] "user_id"                  "status_id"
## [3] "created_at"                "screen_name"
## [5] "text"                      "source"
## [7] "display_text_width"         "reply_to_status_id"
## [9] "reply_to_user_id"           "reply_to_screen_name"
## [11] "is_quote"                  "is_retweet"
## [13] "favorite_count"            "retweet_count"
## [15] "quote_count"               "reply_count"
```

```
## [17] "hashtags"                      "symbols"
## [19] "urls_url"                       "urls_t.co"
## [21] "urls_expanded_url"               "media_url"
## [23] "media_t.co"                      "media_expanded_url"
## [25] "media_type"                      "ext_media_url"
## [27] "ext_media_t.co"                  "ext_media_expanded_url"
## [29] "ext_media_type"                  "mentions_user_id"
## [31] "mentions_screen_name"            "lang"
## [33] "quoted_status_id"                "quoted_text"
## [35] "quoted_created_at"               "quoted_source"
## [37] "quoted_favorite_count"           "quoted_retweet_count"
## [39] "quoted_user_id"                  "quoted_screen_name"

## [41] "quoted_name"                     "quoted_followers_count"
## [43] "quoted_friends_count"             "quoted_statuses_count"
## [45] "quoted_location"                 "quoted_description"
## [47] "quoted_verified"                 "retweet_status_id"
## [49] "retweet_text"                    "retweet_created_at"
## [51] "retweet_source"                  "retweet_favorite_count"
## [53] "retweet_retweet_count"            "retweet_user_id"
## [55] "retweet_screen_name"              "retweet_name"
## [57] "retweet_followers_count"          "retweet_friends_count"
## [59] "retweet_statuses_count"            "retweet_location"
## [61] "retweet_description"              "retweet_verified"
## [63] "place_url"                       "place_name"
## [65] "place_full_name"                 "place_type"
## [67] "country"                         "country_code"
## [69] "geo_coords"                      "coords_coords"
## [71] "bbox_coords"                     "status_url"
## [73] "name"                            "location"
## [75] "description"                     "url"
## [77] "protected"                       "followers_count"
```

```
## [79] "friends_count"           "listed_count"  
## [81] "statuses_count"          "favorites_count"  
## [83] "account_created_at"       "verified"  
## [85] "profile_url"              "profile_expanded_url"  
## [87] "account_lang"              "profile_banner_url"  
## [89] "profile_background_url"    "profile_image_url"
```

Following is a brief description of all the columns returned by Twitter:

Twitter:

Twitter:

Twitter: Twitter:

Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter:

Twitter: Twitter:

Twitter: Twitter:

Twitter: Twitter:

Twitter: Twitter: Twitter:

Twitter: Twitter: Twitter:

Twitter:

Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter:

Twitter: Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter:

Twitter:

Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter:

Twitter: Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter:

Twitter:

Twitter:

Twitter: Twitter: Twitter:

Twitter:

Twitter: Twitter:

Twitter:

Twitter: Twitter:

Twitter: Twitter:

Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter: Twitter:

Twitter:

Twitter: Twitter:

|          |
|----------|
| Twitter: |
| Twitter: |
| Twitter: |

|                   |
|-------------------|
| Twitter:          |
| Twitter: Twitter: |

|          |
|----------|
| Twitter: |

**Table 11.1:** The description1: Description of the columns returned by Twitter using the search\_tweets() function. The values shown in the Example column are as returned by the search\_tweets() function. The values in the Example column are not from the same Tweet.

To see the number of tweets returned by Twitter, we can use the nrow() function shown as follows:

```
nrow(v_temp)  
## [1] 994
```

We can view the tweets by just typing the data frame name of the data frame. However, the display will be very illegible as so much information is returned by Twitter. A much more convenient method of viewing tweets (or any large data frame) is to use the view() function:

```
view(v_temp)
```

A sample output generated using RStudio is shown as follows:

|    | user_id             | status_id           | created_at          | screen_name   | text  |
|----|---------------------|---------------------|---------------------|---------------|---|
| 1  | 3634219694          | 1302048361667727360 | 2020-09-05 00:58:14 | AJOnSchultz   | Experts warn U.S. covid-19 deaths could more        |
| 2  | 46202044            | 1302048352763351041 | 2020-09-05 00:58:12 | ApriaMalita   | New story on NPR: New Global Coronavirus Dea        |
| 3  | 4063933733          | 1302048348598403075 | 2020-09-05 00:58:11 | tomGELP22     | Si tan solo el coronavirus se evitara con un prot   |
| 4  | 1287008070          | 1302048347650392064 | 2020-09-05 00:58:11 | ShannonAnNead | @happyhelent @JaneMortgages @ProfKarolSiko          |
| 5  | 1287008070          | 1302047581850271744 | 2020-09-05 00:55:09 | ShannonAnNead | @happyhelent @JaneMortgages @ProfKarolSiko          |
| 6  | 1105148736873865219 | 1302048343850446848 | 2020-09-05 00:58:10 | ContravisionN | 🇪🇸 España suma 10.476 contagios de coronavirus      |
| 7  | 26387478            | 1302048336564764672 | 2020-09-05 00:58:08 | sharped       | To add to the list of businesses doing good CO      |
| 8  | 101958391           | 1302048336153841666 | 2020-09-05 00:58:08 | javierlanza   | Y mientras Sudamérica está al rojo vivo con tem     |
| 9  | 342405894           | 1302048332248948736 | 2020-09-05 00:58:07 | antoniore10   | #JornaldaCultura Perfeito Dr. Arnaldo, Esse cor     |
| 10 | 14874228            | 1302048328381759494 | 2020-09-05 00:58:07 | santiak       | *Cuatro meses después de que la ciudad portua       |
| 11 | 788504361387364352  | 1302048325051527168 | 2020-09-05 00:58:06 | candleguyn    | Trump approval back up to pre-coronavirus sh        |
| 12 | 14662550            | 130204833483341666  | 2020-09-05 00:58:06 | unreco        | Trump's re-election chances will be weakened by new |

**Figure 11.1:** Output of view data frame containing tweets

## Fetching tweets for a Location

For fetching a tweet for a location, we need to set up the Twitter account just like we did for fetching tweets for a search string. In addition to this, we need to use the Google Maps API Key. Without it, we can only search tweets for a few locations. Using the Google Maps API key, we can search tweets for any location that has been pinned on Google Maps from anywhere in the world.

We need to set up the Google Maps API key. We can read the Google Maps API key from the SystemParameters.csv file shown as follows:

```
##### SET UP THE GOOGLE MAPS
#####
google_maps_api_key =
as.character(v_system_parameters[(which(Parameter ==
"google_maps_api_key")),$Value])
```

We can use the Google Maps API Key to find the coordinates of any location shown as follows:

```
p_search_location <- "London"
```

```
## Locate Location
v_location <- lookup_coords(
```

```
p_search_location,  
apikey = google_maps_api_key  
)
```

The function `lookup_coords()` gets the coordinates of any location. The location can be the name of a country or a city or a street and anything of this nature. Following is the output of the function

```
v_location  
## $place  
## [1] "London"  
##  
## $box  
## sw.lng.lng sw.lat.lat ne.lng.lng ne.lat.lat  
## -0.13333333 51.48333333 -0.03333333 51.58333333  
##  
## $point  
##          lat          lng  
## 51.53333333 -0.08333333  
##  
## attr("class")  
## [1] "coords" "list"
```

Once we have the coordinates of a location, we can fetch tweets from that location using the following code:

```
## Fetch Tweets  
v_temp <- search_tweets(  
geocode = v_location,
```

```
'-filter' = "replies",
n = 1000,
include_rts = FALSE,
retryonratelimit = FALSE
)
```

Notice that we pass the location coordinates to the parameter geocode in the search\_tweets() function.

## Displaying tweets using DT

We need to be able to view the tweets that we have fetched. We know that the matter of a tweet is available in the column text from the data returned by Twitter. We can use a data table to display the Tweets as this data is large in amount tweets. We know from [Chapter 6: Shiny Application 2](#) that we can create a data table using the function datatable() from the DT library. We will also need the dplyr library. So, we need to load the DT and dplyr libraries [we learned how to do this in [Chapter 1: Getting Started with](#) Let us discuss the code required to form the data table:

```
##### DISPLAY FETCHED TWEETS
#####
DT::datatable(
  {select(v_temp, text)},
  colnames = c('Tweet'),
  caption = htmltools::tags$caption(
    style = 'caption-side: bottom; text-align: center;',
    'Table 1: ',
    htmltools::em(paste('Fetched Tweets. Total:', nrow(v_temp)))
  ),
  extensions = 'Buttons',
  escape = TRUE,
  options = list(
    fixedColumns = TRUE,
    autoWidth = FALSE,
```

```
ordering = TRUE,  
pageLength = 15,  
dom = 'Bftsp',  
  
buttons = c('copy', 'csv')  
))
```

You would be familiar with this code as we discussed data tables in [Chapter 6 Shiny Application](#). Let us discuss a few new aspects. Notice that, for the data to be displayed in the data table, we have passed `select(v_temp, text)`. We know that the `select()` function is available in the `dplyr` library. The `select()` function selects one or more columns from a data frame. So, we have selected the column `text` from the data frame. The column `text` contains the contents of the tweet.

Next, notice the parameter `colnames`. Using this parameter, we can specify the names of each column to be displayed in the data table. Also, we can specify the order in which each column has to be displayed in the data table. The parameter `colnames` takes a vector as input.

Lastly, under the parameter options, notice that we have added a new option called `options`. Following is the output generated by the preceding command:

|    |   | Search:                                  |
|----|---|--|
|    |   | <a href="#">Copy</a> <a href="#">CSV</a> |
|    |   | Tweet                                    |
| 1  | @aruzyu_R10 チュートリアルの時点で心臓ががががががが_(3'')  | _上級者に勝つのは大変そうですね...                      |
| 2  | @dK9rsOatH73f0xx ならないように気をつけなよ()  |  |
| 3  | @mnmt_arknights 2つ目読んで私は力尽きた  |  |
| 4  | #私がTwitterにいないと寂しい人はrt メンヘラタグ   |  |
| 5  | @dK9rsOatH73f0xx ノ('; ワ; `)ノセイイイ  |  |
| 6  | @nyanyanun バルキリースカート  |  |
| 7  | 龍門幣の幣の字が弊になってる人多すぎい   |  |
| 8  | @pastimepelusa べるーさんこんグラニ！  |  |
| 9  | 第5人格、心臓がドキドキするタイプのゲームじゃったか...   |  |
| 10 | 自分が好きなやつが好き #Peing #質問箱 <a href="https://t.co/Te2xbhh7fE">https://t.co/Te2xbhh7fE</a>                             |  |
| 11 | @D78890477 なるほど！？ だいぶ前に昇進1にはしたんですが、レベルが低すぎて弱かったんですね..... ちょこちょこ見て見ようかな！   |  |
| 12 | @Mogamin_Lord わかる(わかる)  |  |
| 13 | @Mogamin_Lord ⑤ポケットモンスター ソウルシルバー ⑥遊撃手(Short Stop) ⑦スクリーンセーバー ⑧シークレットサービス 増やしてやるっ                                  |  |
| 14 | @Mogamin_Lord は、半日...   |  |
| 15 | 弊口ドスの秘書はグラニちゃん固定で変更できない仕様になっておりますので、みなさん頑張ってください<br><a href="https://t.co/YBLiTDOw0t">https://t.co/YBLiTDOw0t</a> |  |

Table 1: Fetched Tweets. Total: 999

Previous 1 2 3 4 5 ... 67 Next

**Figure 11.2:** Data table displaying the tweets

Notice that the column name is tweet as specified using the colnames parameter. Also, notice that 15 tweets are displayed on one page as specified for the option You would be familiar with the remaining features of the data table that we discussed in [Chapter 6: Shiny Application](#)

We could make one enhancement to the preceding data table by displaying the tweets with the link to the URL where the tweets are available on Twitter. Notice that in the datatable() function call, we have set the parameter escape to This parameter setting, escape = specifies that any URL, if available in the data table would be inactive and will not result in any URL invocation.

## Displaying tweets with their links

The column status\_url contains the URL where the tweet is available on the internet. We could create a new column in the data frame to create the link to the Tweet. Look at the following code to understand this:

```
v_temp <- v_temp %>%
  mutate(tweet_with_link =
    paste("href='",
          status_url,
          "'",
          "target='_blank'>",
          text,
          "'",
          sep = ""))
```

Notice that we formed a HTML tag. Here are contents of the column text are displayed to the user and the hyperlink is provided from the column Setting target = '\_blank' ensures that when the hyperlink is clicked, a new browser window/tab is opened. We have used the mutate() function from the dplyr library to create a new column named tweet\_with\_link containing these links to the tweets. We could now use the contents of this new column tweet\_with\_link to create the data table shown as follows:

```
DT::datatable(
```

```
{select(v_temp, tweet_with_link)},  
colnames = c('Tweet Text (Click on the Tweet to see the Original  
Tweet)'),  
caption = htmltools::tags$caption(  
  
style = 'caption-side: bottom; text-align: center;',  
'Table 1: ',  
htmltools::em('Fetched Tweets')  
,  
extensions = 'Buttons',  
escape = FALSE,  
options = list(  
fixedColumns = TRUE,  
autoWidth = FALSE,  
ordering = TRUE,  
pageLength = 10,  
dom = 'Bftsp',  
buttons = c('copy', 'csv')  
))
```

Notice that we have set `escape = FALSE` so that the data table allows for external links to be clicked. The output generated is shown as follows:

|   | <a href="#">Copy</a> | <a href="#">CSV</a> | <a href="#">Search:</a>  |
|---|----------------------|---------------------|--|
| Tweet Text (Click on the Tweet to see the Original Tweet) |                      |                     |  |
| 1   |                      |                     | <a href="#">@aruzyu R10 チュートリアルの時点で心臓がががががが (3'4) 上級者に勝つのは大変そうですね...</a> |
| 2   |                      |                     | <a href="#">@dK9rsOatH73f0xx ならないように気をつけなよ()</a>                         |
| 3   |                      |                     | <a href="#">@mnmt_arknights 2つ目読んで私は力尽きた</a>                             |
| 4   |                      |                     | <a href="#">#私がTwitterにいないと寂しい人はメンヘラタグ</a>                               |
| 5   |                      |                     | <a href="#">@dK9rsOatH73f0xx 0(' ッ; ` )ノセイイイ</a>                         |
| 6   |                      |                     | <a href="#">@nyanyanun バルキリースカート</a>                                     |
| 7   |                      |                     | <a href="#">龍門幣の幣の字が弊になってる人多すぎい</a>                                      |
| 8   |                      |                     | <a href="#">@pastimepelusa べるーさんこんグラニ！」</a>                              |
| 9   |                      |                     | <a href="#">第5人格、心臓がドキドキするタイプのゲームじゃったか...</a>                            |
| 10  |                      |                     | <a href="#">自分が好きなやつが好き #Peing #質問箱 https://t.co/Te2xbhh7fE</a>          |

Table 1: Fetched Tweets

Previous [1](#) [2](#) [3](#) [4](#) [5](#) ... [100](#) Next

**Figure 11.3:** Data table displaying tweets with their associated Links

Try clicking on the links in this data table to view the tweets in Twitter.

### Analyzing information obtained from Twitter

Now that we have discussed how to gather data from Twitter, let us start analyzing this data. As the data obtained from Twitter is available in a data frame, it should be easy for you understand the various analyses that we will discuss in this chapter.

For all the discussion on the different analysis, I will assume that you would have fetched the data from Twitter.

## Time Series chart for when tweets were posted

We will now create a Time Series Chart for when the tweets were posted. The Time Series Chart can provide information on the patterns depicting the subject, location, etc. of the Tweets being actively posted tweets. Using we can generate the Time Series Charts for intervals of seconds, minutes, or hours.

We know that the column created\_at contains the time at which the tweet was posted. Before we can create the Time Series Chart, we need to convert the value in the column created\_at to the POSIXct format as shown in the following code:

```
# Save current locale
loc <- Sys.getlocale("LC_TIME")

# Set Locale
Sys.setlocale("LC_TIME", "C")

# Convert to POSIXct
v_temp$created_at <- as.POSIXct(v_temp$created_at, '%Y-%m-%d
%H:%M:%S', tz = Sys.timezone())

# Then set back to the old locale
Sys.setlocale("LC_TIME", loc)
```

The preceding code should be self-explanatory. Once we have converted the value of the column created\_at to the POSIXct format, we can use the same for creating the Time Series Chart using the ts\_plot() function of the ggplot2 library as shown in the following code. You need to load the ggplot2 library [we have discussed this in [Chapter 1: Getting Started with](#)

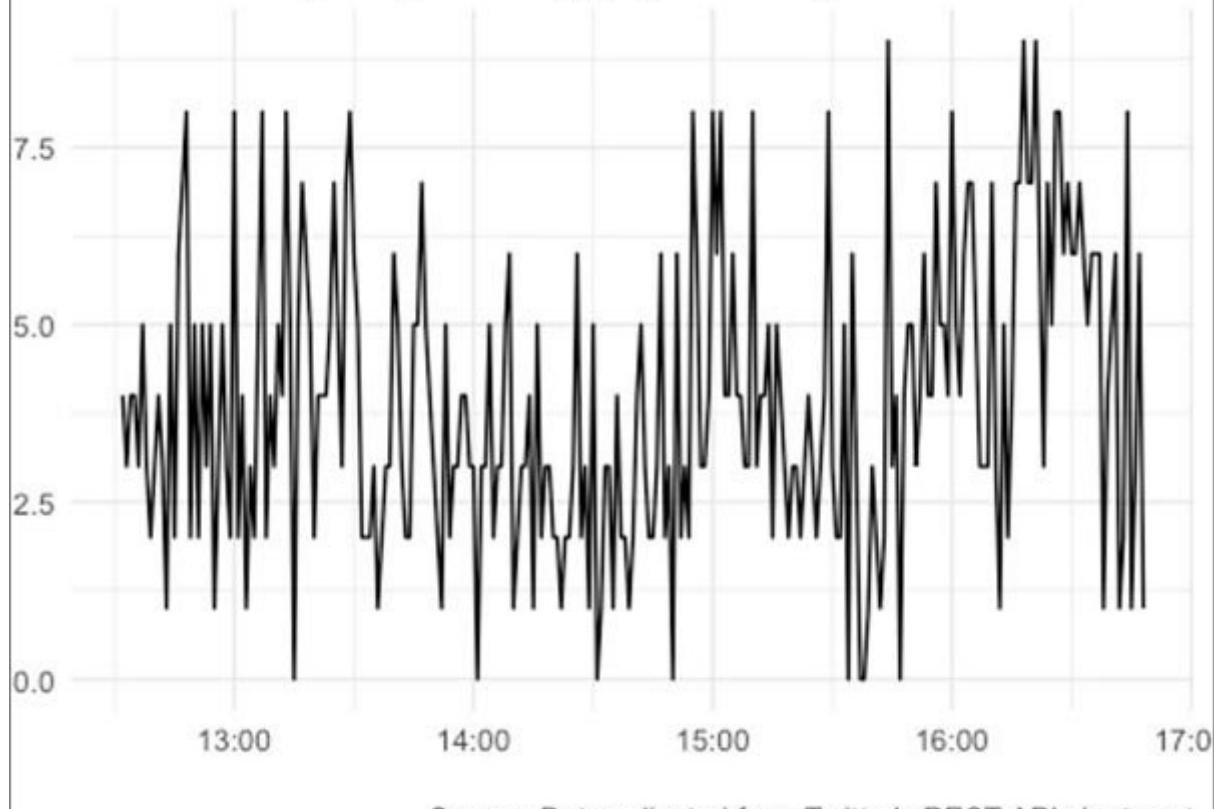
```
## Plot time series of tweets frequency

ts_plot(v_temp[, c("created_at")], "mins") +
  ggplot2::theme_minimal() +
  ggplot2::theme(plot.title = ggplot2::element_text(face = "bold")) +
  ggplot2::labs(
    x = NULL, y = NULL,
    title = paste("Frequency of", p_search_location, "Tweets statuses",
      sep = " "),
    subtitle = "Twitter status (tweet) counts aggregated using one-
      minute intervals",
    caption = "\nSource: Data collected from Twitter's REST API via
      rtweet"
  )
```

Note that we pass only the column created\_at (converted to the POSIXct format) from the data frame v\_temp to the ts\_plot() function. The output from this code could be as shown in the following figure:

## Frequency of Paris Tweets statuses

Twitter status (tweet) counts aggregated using one-minute intervals



**Figure 11.4:** Time Series Chart for tweets from Paris at a certain point of time

## Country analysis

We can analyze the countries from where people have tweeted during a certain point of time. This is a basic analysis. This can have many extensions. I leave that to your imagination and innovation. We know that the columns country and country\_code contain the data regarding the country and the associated country code respectively from where the tweet was posted. We can use this data for analyzing the countries from where Twitter users have posted their tweets.

After having fetched the tweets, we can create a factor of the country of the tweets using the following code:

```
v_temp_df <- v_temp %>%
ungroup() %>%
mutate(fac_country = as.factor(country))
```

Now, if we check the unique values of the country of the tweets, then we will get the following results:

```
unique(v_temp_df$country)
## [1] NA          "Spain"      "United
States" "Canada"
## [5] "India"      "Brazil"
"Chile"     "Qatar"
## [9] "United Kingdom" "Georgia"    "Kenya"
```

Notice that there is an entry for NA. The presence of NA in the list indicates that there are records where the value in the country column is missing.

In this circumstance, if we try to find the frequency table of the countries based on the number of tweets, we would get the following result:

```
v_country <- as.data.frame(  
  table(v_temp_df$fac_country))  
)  
  
v_country  
##          Var1 Freq  
## 1        Brazil    1  
## 2      Canada    1  
## 3       Chile    1  
## 4     Georgia    1  
## 5       India    2  
## 6       Kenya    1  
## 7       Qatar    1  
## 8       Spain    3  
## 9 United Kingdom    2  
## 10 United States   3
```

Notice that the frequency table does not contain any entry for the missing values. This will cause a problem for any application that we would write around this data. This is because it is very much possible that all the values in the country column are missing in the data that we fetch.

To avoid this pitfall, we need to use the `fct_explicit_na()` function from the `forcats` library shown as follows:

```
v_country <- as.data.frame(  
  table(  
    fct_explicit_na(v_temp_df$fac_country)  
)  
)
```

```

## Display Result

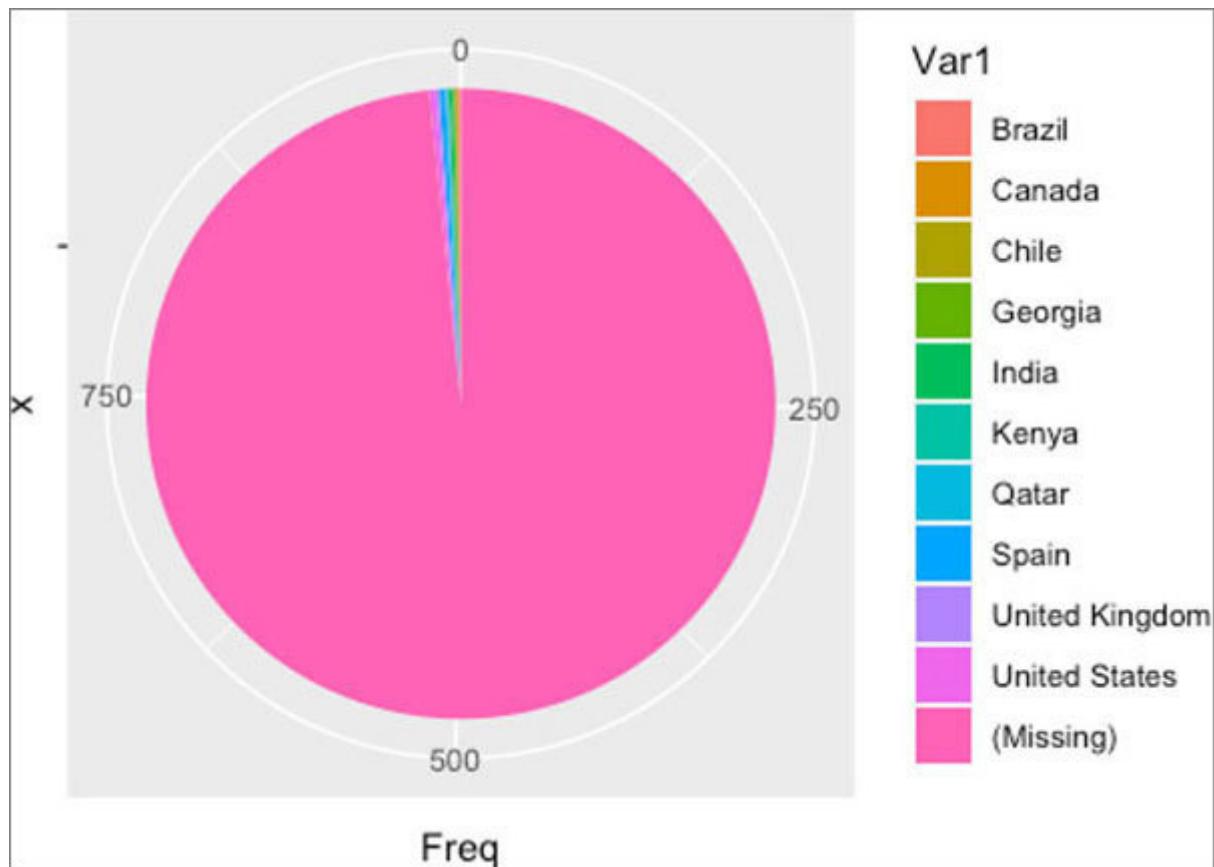
v_country
##          Var1 Freq
## 1        Brazil   1
## 2      Canada   1
## 3       Chile   1
## 4    Georgia   1
## 5       India   2
## 6       Kenya   1
## 7       Qatar   1
## 8       Spain   3
## 9 United Kingdom   2
## 10 United States  3
## 11 (Missing) 979

```

Notice that now we have an explicit entry in the frequency table for the missing values. Let us create a pie chart based on this frequency table:

```
bp<- ggplot(v_country, aes(x="", y=Freq, fill=Var1)) +  
  geom_bar(width = 1, stat = "identity")  
pie <- bp + coord_polar("y", start=0)  
pie
```

The following is the output of the preceding code:

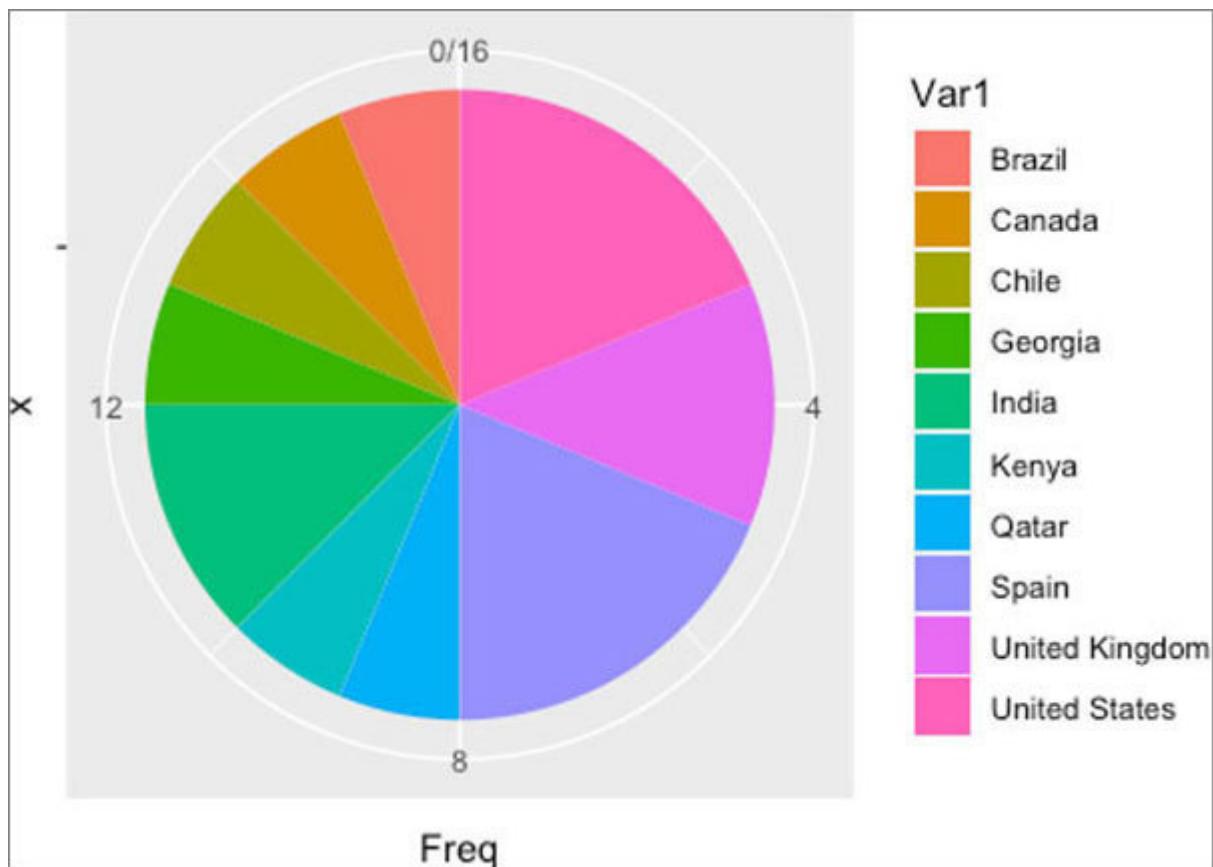


**Figure 11.5:** Pie chart based on the frequency table of countries from where tweets were posted

We notice that this pie chart is not very useful as more than 95% of the data is that of the missing values. So, we create a pie chart by excluding the missing values:

```
bp<- ggplot(v_country[which(v_country$Var1 != '(Missing)'),],  
aes(x="", y=Freq, fill=Var1)) +  
geom_bar(width = 1, stat = "identity")  
pie <- bp + coord_polar("y", start=0)  
pie
```

The following is the output:



**Figure 11.6:** Pie chart based on the frequency table of countries from where tweets were posted excluding the missing data

## Place analysis

We can conduct place analysis very similar to the way we conducted country analysis. The column place\_full\_name contains the name of the place from where the user posted the tweet. So, we can use the following code:

```
v_temp_df <- v_temp %>%
ungroup() %>%
mutate(fac_Place = as.factor(place_full_name))

## Compute unique values for Place Code
v_place <- as.data.frame(
table(
fct_explicit_na(v_temp_df$fac_Place)
)
)

## If no Place found, create an empty data frame
if(nrow(v_place) <= 0) {
v_place <- data.frame(Var1 = character(),
Freq = integer(),
stringsAsFactors = FALSE)
}
```

The following is the output::

```
## Display Result
```

```

v_place
##                                     Var1 Freq
## 1                               Ascot, South East    1
## 2                               Carapicuíba, Brasil    1
## 3 Ciudad Autónoma de Buenos Aires, Argentina    1
## 4                               Claymont, DE    1

## 5                Johannesburg, South Africa    1
## 6           Las Palmas de Gran Canaria, Spain    1
## 7                   Long Beach, CA    1
## 8               Montréal, Québec    1
## 9            Napili-Honokowai, HI    1
## 10                  Osasco, Brasil    1
## 11                 Quilmes, Argentina    1
## 12                 Sao Paulo, Brazil    1
## 13                  Temple, TX    1
## 14          (Missing)    987

```

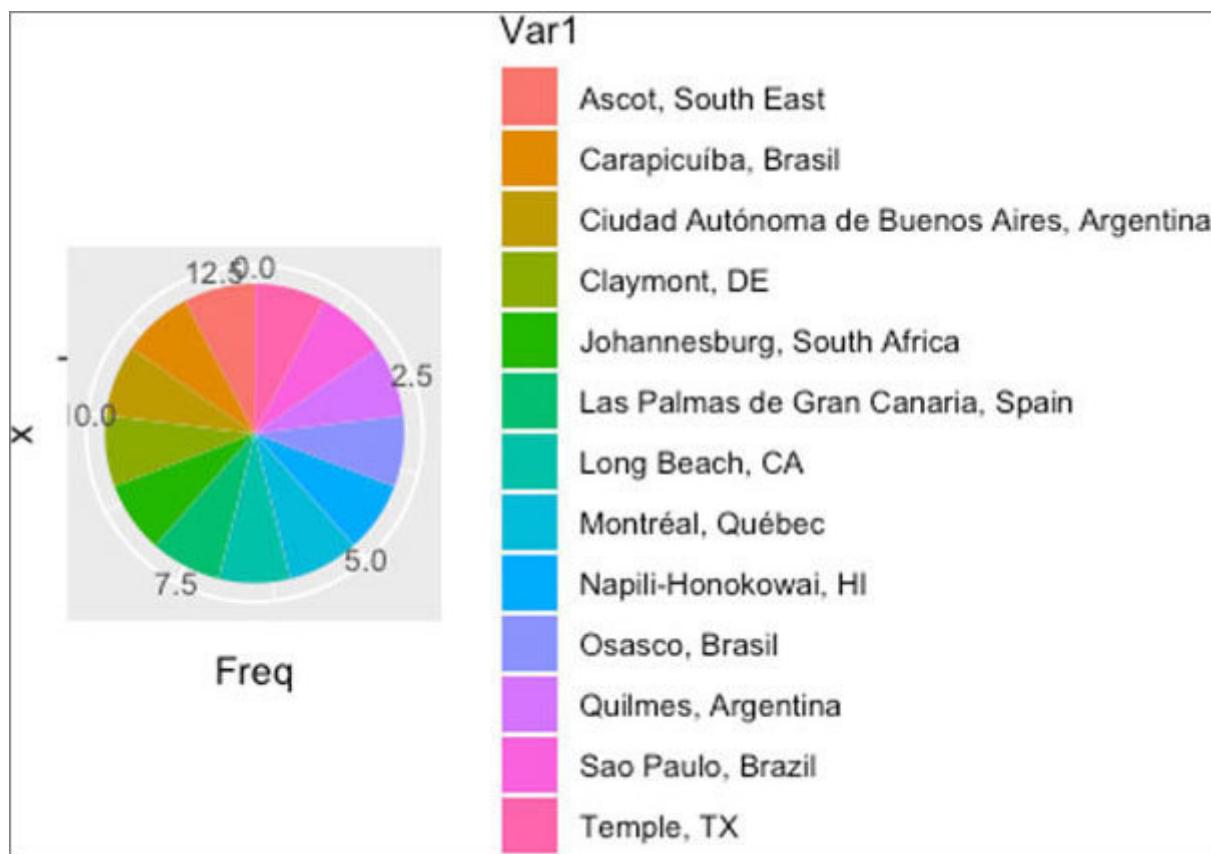
Now, we can create the following pie chart:

```

bp<- ggplot(v_place[which(v_place$Var1 != '(Missing)'),],
aes(x="", y=Freq, fill=Var1)) +
geom_bar(width = 1, stat = "identity")
pie <- bp + coord_polar("y", start=0)
pie

```

The following is the output::



**Figure 11.7:** Pie chart based on the frequency table of places from where tweets were posted excluding the missing data

## Language analysis

Language analysis can be done similar to the country and place analysis. The column **lang** contains the language in which the tweet has been posted. So, we can use the following code to generate the frequency table of the language of the tweet:

```
v_temp_df <- v_temp %>%
  ungroup() %>%
  mutate(fac_language = as.factor(lang))

## Compute unique values for Language Code
v_language <- as.data.frame(
  table(
    fct_explicit_na(v_temp_df$fac_language)))
```

The frequency table generated is shown as follows:

```
v_language
##     Var1 Freq
## 1    ar    6
## 2    bn    1
## 3    ca   55
## 4    cy    1
## 5    de   11
## 6    en  499
## 7    es  236
```

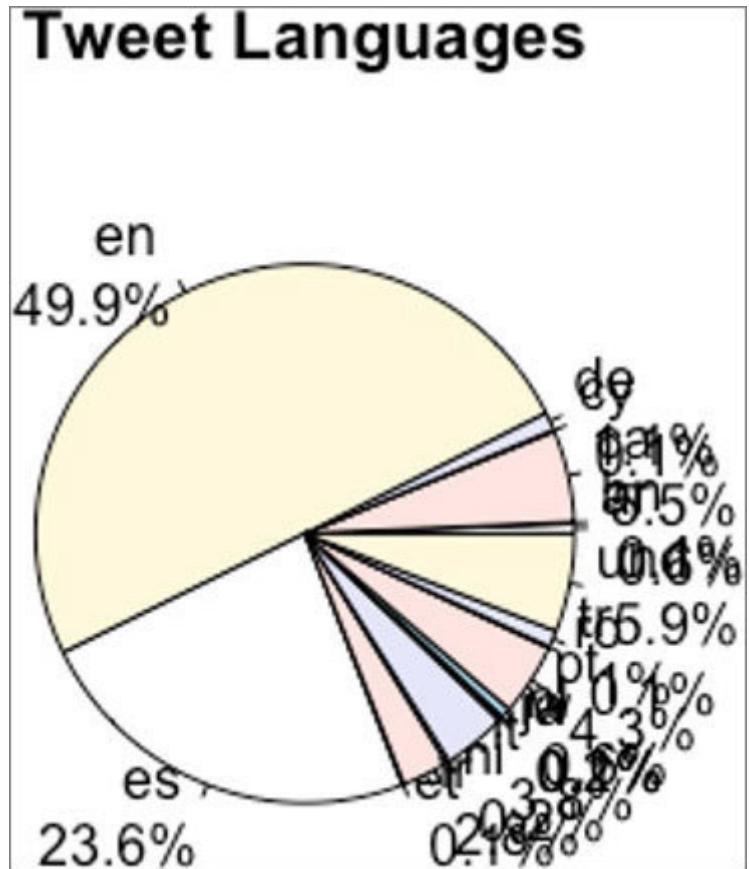
```
## 8     et     1
## 9     fr    28
## 10    hi     2
## 11    it    38
## 12    iw     2

## 13    ja     1
## 14    nl     6
## 15    pt    43
## 16    ro     1
## 17    tr    10
## 18    und    59
```

Notice that there is no entry for missing data in this frequency table. This implies that there was no missing data for this column. We can create a pie chart using the `pie()` function shown as follows:

```
v_data <- v_language$Freq
v_labels <- paste(paste(v_language$Var1, "\n",
round(v_data/sum(v_data)*100, 2), sep=" "),
"%", sep = "")  
pie(v_data, labels = v_labels, main = "Tweet Languages")
```

The following is the output of this code:



**Figure 11.8:** Pie chart based on the frequency table of the languages used in the tweets

## User analysis

We will now discuss how to conduct an analysis of the Twitter users who have posted the tweets. The column name contains the name of the Twitter user who posted the tweet. The columns followers\_count and friends\_count contain the number of followers and friends of the Twitter user.

We can create the frequency table of the number of tweets posted by the Twitter user using the following code:

```
## Convert User Codes to a factor  
v_temp_df <- v_temp %>%  
ungroup() %>%  
mutate(fac_User = as.factor(name))  
## Compute unique values for User Code  
v_user <- as.data.frame(  
table(  
fct_explicit_na(v_temp_df$fac_User)  
)  
)
```

For each Twitter user in this list, we want to attach the number of followers and friends of the Twitter users. We can use the following code for this purpose:

```
## Determine Friends and Followers Count
```

```

if(nrow(v_user) > 0) {
v_user <- v_user %>%
mutate(Tweeter = reorder(Var1, Freq)) %>%
filter(Tweeter != "(Missing)")

# Determine the number of Followers and Friends

v_user$Followers <- as.integer(0)
v_user$Friends <- as.integer(0)
for (i in 1:nrow(v_user)) {
v_user[i,]$Followers <- as.integer(mean(subset(v_temp_df, fac_User
== v_user[i,]$Tweeter)$followers_count))
v_user[i,]$Friends <- as.integer(mean(subset(v_temp_df, fac_User ==
v_user[i,]$Tweeter)$friends_count))
}
} else {
v_user <- data.frame(Tweeter = character(),
Freq = integer(),
Followers = integer(),
Friends = integer(),
stringsAsFactors = FALSE)
}

```

Now, we can display this data frame using DT shown as follows:

```

## Sort Result
attach(v_user)
v_user <- v_user[order(-Freq),]
DT::datatable(
{v_user[,c("Tweeter", "Freq", "Followers", "Friends")]}, 
colnames = c("Tweeter", "#", "Followers", "Friends"),

```

```

escape = TRUE,
rownames = FALSE,
options = list(
fixedColumns = TRUE,
autoWidth = FALSE,
ordering = TRUE,

```

```

pageLength = 10,
dom = 'tip'
))

```

Notice that we displayed more than one column in the data table and ordered the columns as per our choice. Also, we have given names to the columns as per our choice. Notice the parameter `rownames = FALSE` which hides the record numbers in the first column from the display. We have not used any extension and thus, the buttons are not displayed. The following is the output produced:

| Tweeter                        | # | Followers | Friends               |
|--------------------------------|---|-----------|-----------------------|
| Ray Giles                      | 7 | 158       | 2                     |
| Donald J. Trump                | 6 | 66        | 8                     |
| Leadership Masters             | 4 | 2812      | 668                   |
| *Not Michael Wilson            | 3 | 245       | 674                   |
| DarkShadows                    | 3 | 17549     | 17547                 |
| democracylives                 | 3 | 58        | 127                   |
| Lee Ivory                      | 3 | 2280      | 4041                  |
| ((Kevin Kalmes)))              | 2 | 4916      | 5405                  |
| Amy                            | 2 | 937       | 353                   |
| André C. Gauthier🌈             | 2 | 1857      | 4991                  |
| Showing 1 to 10 of 955 entries |   | Previous  | 1 2 3 4 5 ... 96 Next |

**Figure 11.9:** Data table showing the list of users who have tweeted in this set of data obtained from Twitter

## Source analysis

Like country analysis and place analysis, we can conduct source analysis. There are many interesting facts that emerge from this analysis. For example, generally, this analysis confirms that most mobile phone users in India use Android phone and most mobile phone users in USA use iPhones.

The column source contains the data regarding the device/medium used to post the tweet. After fetching the tweets, we can create the frequency table of the source used to post the Tweet as shown in the following code:

```
## Convert Source Codes to a factor  
v_temp_df <- v_temp %>%  
ungroup() %>%  
mutate(fac_Source = as.factor(source))
```

```
## Compute the Frequency Table  
v_source <- as.data.frame(  
table(  
fct_explicit_na(v_temp_df$fac_Source)  
)  
)
```

We can see a part of the result in the following code:

```
## Display Result
attach(v_source)
v_source <- v_source[order(-Freq),]
head(v_source, 10)
##          Var1 Freq
## 62      Twitter Web App 325
## 57 Twitter for Android 242

## 59 Twitter for iPhone 196
## 56          TweetDeck 33
## 17          dlvr.it 27
## 58 Twitter for iPad 16
## 65 WordPress.com 14
## 7           Buffer 12
## 29 Hootsuite Inc. 11
## 26 GlobalPandemic.NET 9
```

## Hashtag analysis

Analyzing the hashtags in the tweets can give us an idea of the topics being discussed by the Twitter users. As a vast population of the world use Twitter, analyzing hashtags provides an idea of the hot topics of discussion at any point of time.

**Before travelling to any destination, I always conduct hash tag analysis of the latest tweets for that location. This gives me an idea of what the people in that location are presently discussing – whether they are happy about something or whether they are worried about something.**

Hashtag analysis is best conducted by creating a word cloud of the hashtags. The column hashtags contains the hashtags used in the tweet. More than one hashtag may be used in a tweet.

We can use the following code below to generate the hashtag frequency table:

```
## Hashtag Analysis  
v_corpus <- Corpus(VectorSource(v_temp$hashtags))  
  
# Create the Term Document Matrix after cleaning the Corpus  
v_tdm <- TermDocumentMatrix(v_corpus,  
control =  
list(removePunctuation = TRUE,
```

```

stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf))
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))

```

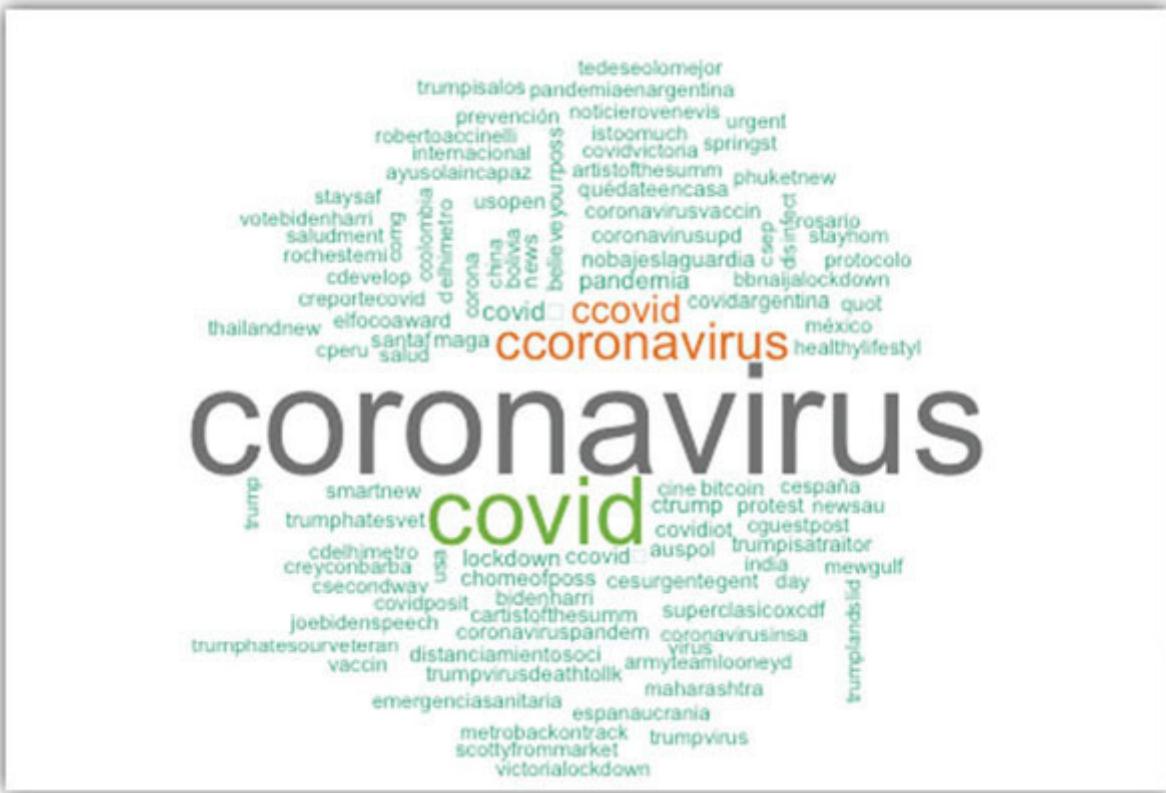
Having generated the frequency table, we can now generate the word cloud shown as follows:

```

wordcloud(words = v_data$ind, freq = v_data$values,
min.freq = 1,
max.words = 100,
random.order=FALSE,
colors=brewer.pal(8, "Dark2")
)

```

The following is the output:



**Figure 11.10:** Hashtag Word Cloud

## Location analysis with maps

Twitter provides location information. This information may be not available for many tweets. However, from the data available, it makes an interesting analysis. Let us discuss how to plant on a map the locations from where people are tweeting at any point of time.

Location information of the tweets is available in three columns – and These three columns contain the data regarding the coordinates of the location from where the tweets have been posted.

First, we need to convert the location information into latitude and longitude. We can do this using the `lat_lng()` function in the `rtweet` library. The `lat_lng()` can use any of the three information, that is, or to determine the latitude and longitude of a location. Unless it is specified, the `lat_lng()` function first considers the data, if available, in the column `bbox_coords` for determining the latitude and longitude. To determine the latitude and longitude of the location from the user who posted the tweet, we can use the following code:

```
v_temp <- lat_lng(v_temp)
```

On executing this command, two new columns are added to the output data frame. These are named `lat` and `lng` to contain the

latitude and longitude of the location, respectively. We can now use the information in these columns to plot it on a map.

But first we need to create the map. We can draw a map using the `map()` function in the `maps` library. To draw the world map, we can issue the following command:

```
library(maps)
map("world")
```



command: command: command: command: command: command:  
command: command: command: command: command: command:  
command: command:

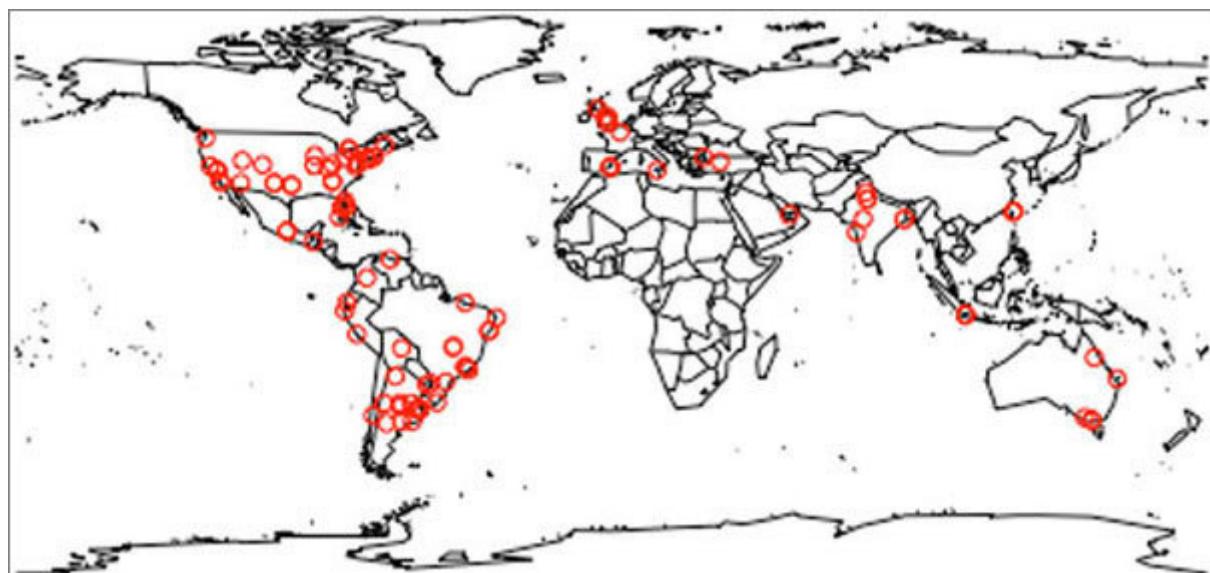
**Table 11.2:** Code for generating the World Map and the Output

After created the map, we can plot the latitudes and longitudes of the locations of the tweets using the following code:

```
## Plot latitude and longitude points onto the World Map
with(v_temp,
points(
lng, lat,
pch = 21,
cex = .75,
col = rgb(1, 0, 0, 1)
)
)
```

We plot the points using the values in lat and lng columns. The pch parameter is used to define the character to be used for marking the point. The cex parameter is used to define the size of the marker. Using the col parameter, we can set the color of the marker.

The output of the preceding command for a set of fetched tweets is shown as follows:



**Figure 11.12:** Locations of a set tweets on coronavirus marked on the World map

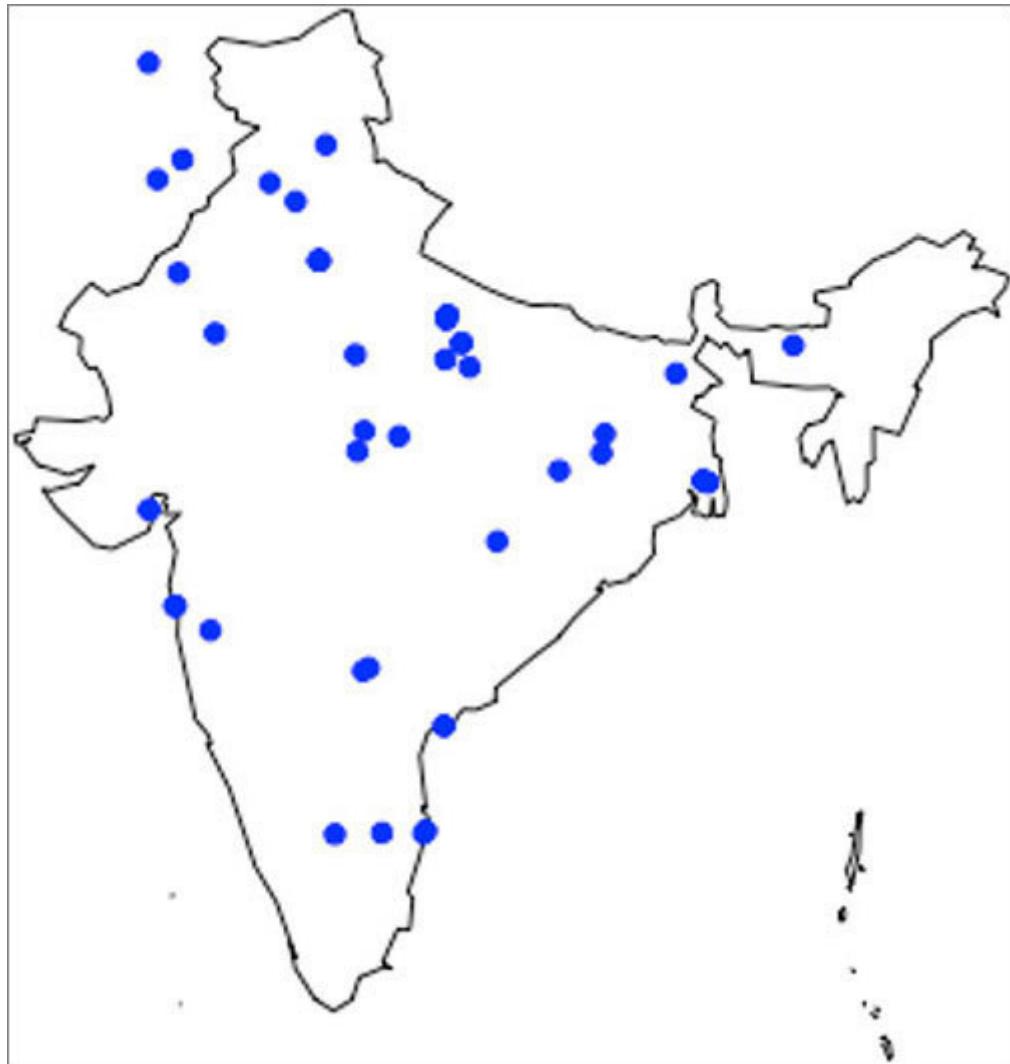
We can create specific maps using the maps library. We could create maps for a country. For example, we can fetch the tweets originating from India and plot them on the map of India. We could do this using the following code (we already discussed how to fetch tweets for a location and so I will not repeat the code):

```
v_temp <- lat_lng(v_temp)
```

```
## Plot the India Map
par(mar = c(0,0,0,0))
map('world', "india",
fill = FALSE,
col = 1:10,
wrap = c(-180,180)
)

## Plot latitude and longitude points onto the India Map
with(v_temp,
      points(
        long, lat,
        pch = 19,
        cex = .75,
        col = rgb(0, 0, 1, 1)
      )
    )
```

The following is the output of this code:



**Figure 11.13:** Locations of a set tweets for the location India marked on the Indian map

## Interactive maps using a leaflet

We can create interactive maps using the leaflet() function in the leaflet library as shown in the following code:

```
#####          FETCH TWEETS FOR SEARCH
STRING #####
v_temp <- search_tweets(
q = "Coronavirus",
'-filter' = "replies",
n = 5000,
include_rts = FALSE,
retryonratelimit = FALSE #,
)

#####
# PLOT TWEET LOCATION ON MAP
#####
v_temp <- lat_lng(v_temp)

## Plot the World Map
leaflet(v_temp) %>%
addCircles(lng = ~lng, lat = ~lat) %>%
addTiles() %>%
addCircleMarkers(data = v_temp, lat = ~lat, lng = ~lng,
radius = 3,
popup = ~as.character(text),
stroke = FALSE, fillOpacity = 0.8)
```



**Figure 11.14:** Output of the map created by the `leaflet()` function of the `leaflet` library

## Emotion analysis of tweets

We now come to the crux of this book, that is, book to extract the emotions from a set of tweets. This is useful as we can get to understand what the people in a city or a country are thinking about in general and/or about a subject. There can be several uses of this information. For regular citizen like you and me, it can maybe help in planning a visit to a city or a country. Or it can help someone staying abroad understand whether any measures need to be taken for the well-being of his/her loved ones in a given circumstance. It can help administrators plan any precautionary measures, mitigation actions, or contingency measures. Or it can help an artist understand whether a new release is positioned appropriately. Eminent people use this information to understand the public sentiment before taking certain a decision. The applications of this information are just countless.

Let us go step-by step in understanding how to extract emotions for a set of tweets. We will apply the concepts and techniques of emotion analysis discussed in [Chapter 8: Emotion](#). Besides, we will use all the knowledge of R programming discussed throughout this book.

We will start by fetching 7,000 tweets on the subject. This can be accomplished using the following code. I assume that you know how to include the necessary libraries:

```

#####      FETCH TWEETS FOR SEARCH
STRING      #####
v_temp <- search_tweets(
q = "Coronavirus",
'-filter' = "replies",

n = 7000,
include_rts = FALSE,
retryonratelimit = FALSE
)

```

The first thing we need to do after fetching the tweets is to extract words from the tweets. Before we can extract the words, we need to clean the data as it may contain URLs, emoticons, etc., which are not useful for our analysis.

The following code removes the emoticons from the fetched data:

```

# Remove the Emoticons
v_temp <- gsub("[^\x01-\x7F]", "", v_temp)

```

Next, we will form the corpus shown as follows:

```

# Form the Corpus
v_corpus <- VCorpus(VectorSource(v_temp))

```

We will remove all the URLs from the corpus shown as follows:

```

# Remove URLs
removeURL <- content_transformer(function(x) gsub(
  "(f|ht)tp(s?)://\\S+", "", x, perl=T))
toSpace <- content_transformer(function (x, pattern) gsub(pattern, " ", x))

v_corpus <- tm_map(v_corpus, removeURL)
v_corpus <- tm_map(v_corpus, toSpace, "/")
v_corpus <- tm_map(v_corpus, toSpace, "@")
v_corpus <- tm_map(v_corpus, toSpace, "\\|")
v_corpus <- tm_map(v_corpus, toSpace, "'")

v_corpus <- tm_map(v_corpus, toSpace, "")
```

We will create two functions, namely removeURL() and toSpace() for this purpose. Both these functions match the patterns and replace the patterns with the replacement. We will use the text mining function tm\_map() available in the tm library and apply these two functions to remove all the URLs in the corpus.

Now, we have the corpus which can be used to create the word frequency shown as follows:

```

# Create the Term Document Matrix after cleaning the Corpus
v_tdm <- TermDocumentMatrix(v_corpus,
control =
list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
```

```

bounds = list(global = c(1, Inf))
)
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame

v_words <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))

```

We can view the word frequency using the `datatable()` function from the DT library:

```

DT::datatable(
  v_words[,c("ind", "values")],
  colnames = c("Word", "#"),
  escape = TRUE,
  rownames = FALSE,
  options = list(
    fixedColumns = TRUE,
    autoWidth = FALSE,
    ordering = TRUE,
    pageLength = 10,
    dom = 'tip'
))

```

| Word        | #     |
|-------------|-------|
| fails       | 26638 |
| cna         | 20854 |
| coronavirus | 6755  |
| twitter     | 4891  |
| covid       | 2738  |
| app         | 2513  |
| web         | 2447  |
| news        | 2174  |
| photo       | 1805  |
| true        | 1545  |

Showing 1 to 10 of 50,797 entries

Previous 1 2 3 4 5 ... 5080 Next

**Figure 11.15:** Data table for viewing the word frequency of words extracted from a set of tweets

We will set up the lexicon and the constants. We have discussed this in [Chapter 8: Emotion](#)

```
v_nrc <- get_sentiments("nrc")
v_positive_emotions <- c("positive", "joy", "anticipation", "surprise",
"trust")
v_negative_emotions <- c("negative", "anger", "disgust", "fear",
"sadness")
v_positive_sentiment <- v_nrc %>%
filter(sentiment %in% v_positive_emotions)
```

Next, we will extract the sentiments available in the tweets shown as follows:

```
v_combined <- sort(union(levels(v_words$word),
levels(v_nrc$word)))
v_text_sentiment2 <- inner_join(
```

```
mutate(v_words, word=factor(word, levels=v_combined)),  
mutate(v_nrc, word=factor(word, levels=v_combined))  
)
```

We will now separate the positive and the negative sentiments. And we will spread the emotions into separate columns – one for each emotion:

```
v_text_sentiment1 <- v_text_sentiment2 %>%  
count(word, index = row_number() %/% 80, sentiment) %>%  
mutate(v_original_n = n) %>%  
mutate(v_positive_negative = ifelse(sentiment %in%  
v_positive_emotions,  
'positive', 'negative')) %>%  
mutate(v_original_sentiment = sentiment) %>%  
spread(sentiment, n, fill = 0)
```

Next, we will ensure that all the emotions in our list are available in our data frame as per the lexicon (that is, Positive, Negative, Joy, Anticipation, Surprise, Trust, Anger, Disgust, Fear, Sadness). Positive, negative, joy, anticipation, surprise, trust, anger, disgust, fear, and sadness) data frame:

```
v_text_sentiment <- v_text_sentiment1 %>%  
mutate(positive = ifelse("positive" %in%  
colnames(v_text_sentiment1), positive, 0)) %>%  
mutate(joy = ifelse("joy" %in% colnames(v_text_sentiment1), joy,  
0)) %>%  
mutate(surprise = ifelse("surprise" %in%  
colnames(v_text_sentiment1), surprise, 0)) %>%
```

```

mutate(trust = ifelse("trust" %in% colnames(v_text_sentiment1),
trust, 0)) %>%
mutate(anticipation = ifelse("anticipation" %in%
colnames(v_text_sentiment1), anticipation, 0)) %>%
mutate(negative = ifelse("negative" %in%
colnames(v_text_sentiment1), negative, 0)) %>%
mutate(anger = ifelse("anger" %in% colnames(v_text_sentiment1),
anger, 0)) %>%
mutate(disgust = ifelse("disgust" %in%
colnames(v_text_sentiment1), disgust, 0)) %>%
mutate(sadness = ifelse("sadness" %in%
colnames(v_text_sentiment1), sadness, 0)) %>%
mutate(fear = ifelse("fear" %in% colnames(v_text_sentiment1), fear,
0)) %>%
mutate(sentiment = (positive+joy+surprise+trust+anticipation) -
(negative+anger+disgust+sadness+fear)) %>%
ungroup()

```

We can determine the contribution of each word to the different emotions:

```

## Determine Words contributing to Emotions
v_data <- v_text_sentiment %>%
filter(row_number() < 50) %>%
group_by(v_original_sentiment) %>%
ungroup() %>%
mutate(word = reorder(word, v_original_n))
v_data$word <- as.character(v_data$word)
v_data <- data.frame(apply(v_data, 2, unclass))

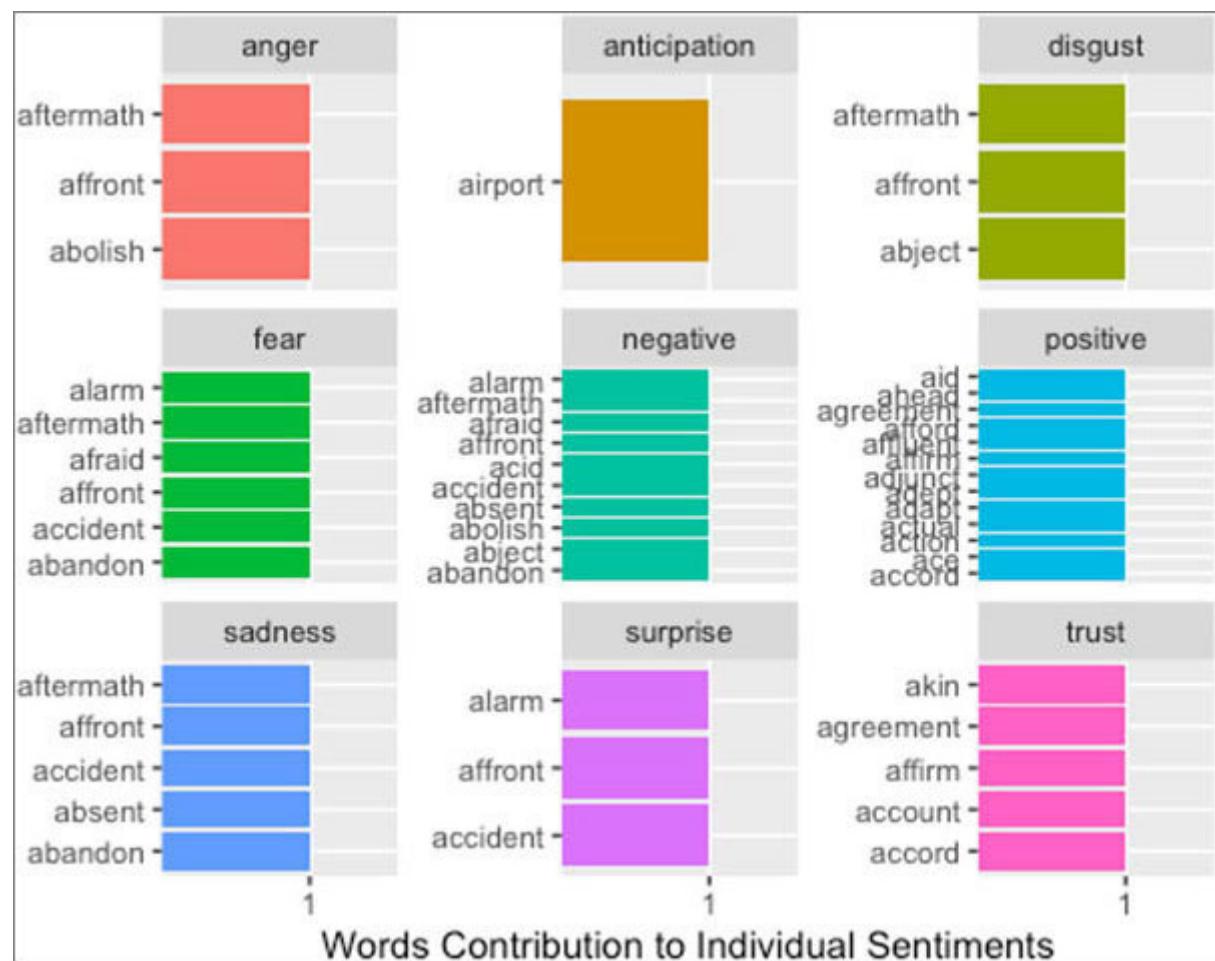
```

```

ggplot(aes(x = word, y = v_original_n, fill = v_original_sentiment),
       data = v_data, scale_x_continuous(breaks = NULL)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~v_original_sentiment, scales = "free_y") +
  labs(y = "Words Contribution to Individual Sentiments", x = NULL)
+
  coord_flip()

```

The following is the output:



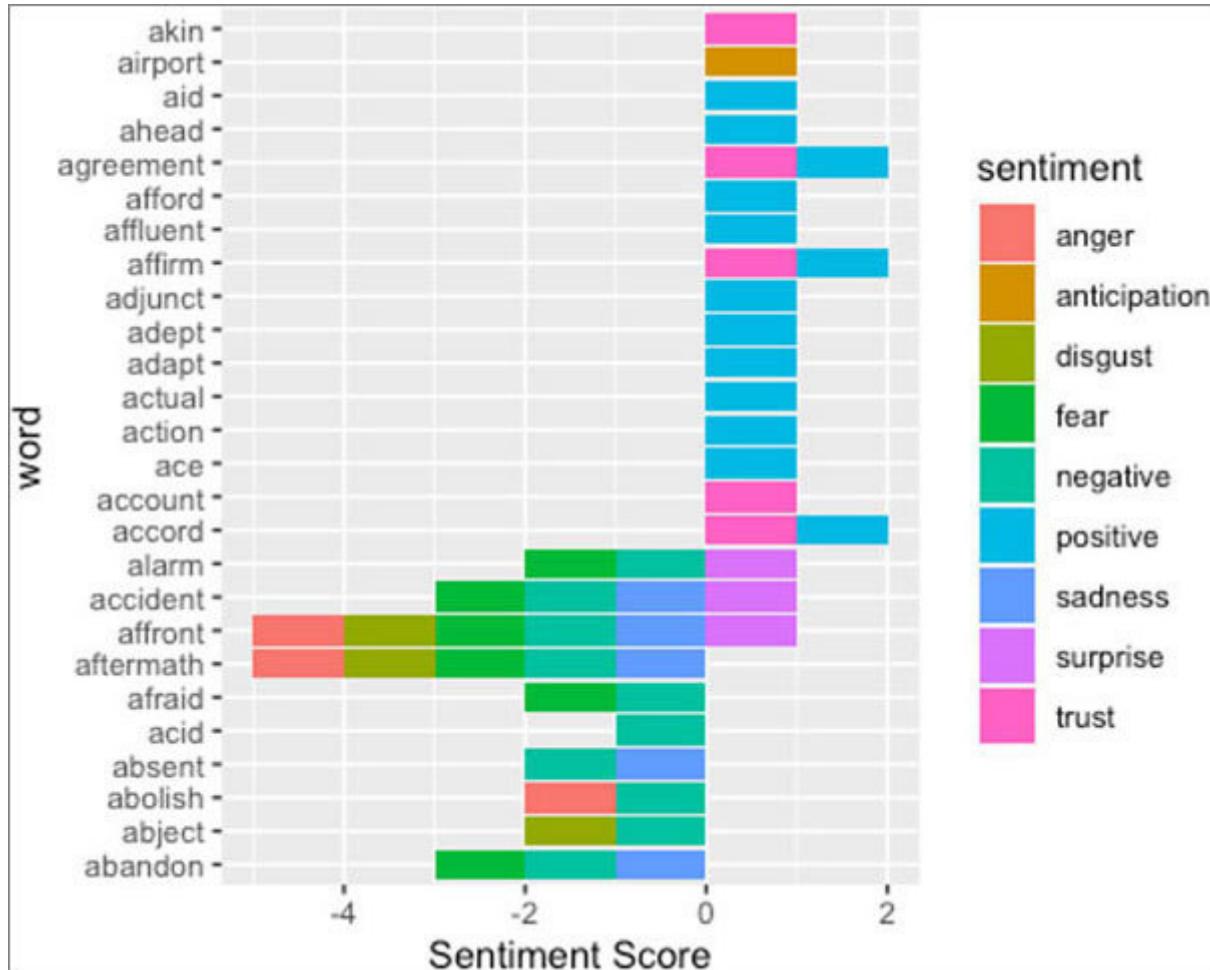
**Figure 11.16:** Word contribution to sentiments for the words found in the tweets

We will create a visualization of the different sentiments emoted by the different words in different contexts:

```
v_data <- v_text_sentiment2 %>%
  filter(row_number() < 50) %>%
  mutate(n = ifelse(sentiment %in% v_negative_emotions, -n, n))
%>%
  mutate(word = reorder(word, n))

ggplot(aes(word, n, fill = sentiment), data = v_data) +
  geom_col() +
  coord_flip() +
  labs(y = "Sentiment Score")
```

The visualization is shown in the following figure. We display all the positive emotions positive emotions on the right and all the negative emotions to the left. As we saw in [Chapter 8: Emotion](#) the same word can emote different emotions in different contexts. Also, the same word can emote both positive and negative emotion in different contexts:



**Figure 11.17:** Sentiment score of each word found in the tweets

Now, we are ready to determine the most prevalent emotion found in the fetched tweets. First, we will present the most prevalent emotion as a text that you can see text as in the following code:

```
v_temp <- v_by_emotions %>%
  mutate_if(is.factor, as.character)

v_prevalent_emotion <- ifelse(nrow(subset(v_temp, values ==
  max(values))) == 1,
  unname(unlist(subset(v_temp, values == max(values)))),
  "No Prevalent Emotion")
```

```
v_prevalent_emotion
```

```
## [1] "fear"
```

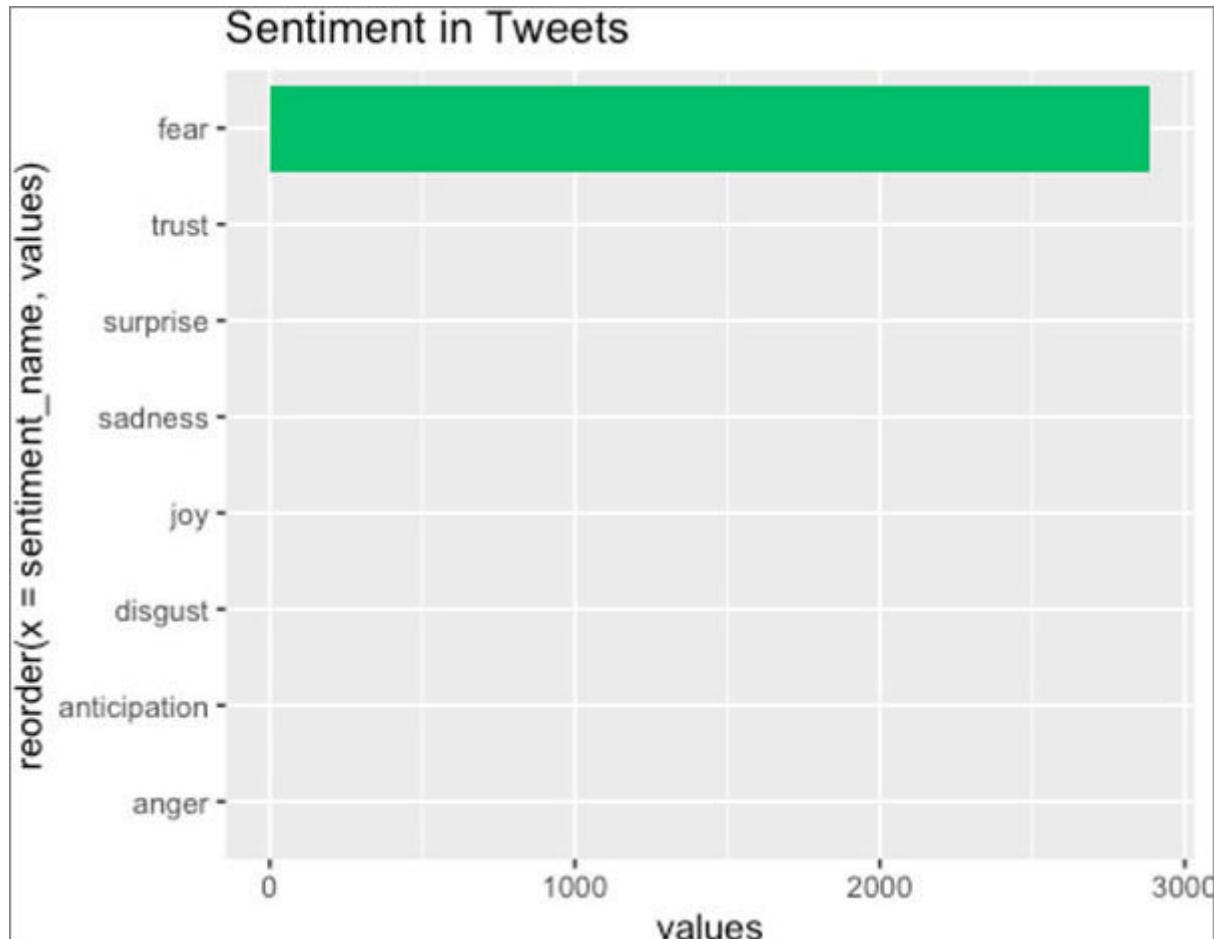
We can present the same graphically as shown in the following code:

```
## Determine Most Prevalent Emotion
v_emotions <- v_text_sentiment %>%
  select(index, anger, anticipation, disgust, fear, joy,
         sadness, surprise, trust) %>%
  melt(id = "index") %>%
  rename(linenumber = index, sentiment_name = variable, value =
         value)
```

```
v_emotions_group <- group_by(v_emotions, sentiment_name)
v_by_emotions <- summarise(v_emotions_group,
                           values=sum(value))
```

```
ggplot(aes(reorder(x=sentiment_name, values), y=values,
           fill=sentiment_name),
       data = v_by_emotions) +
  geom_bar(stat = 'identity') +
  ggtitle('Sentiment in Tweets') +
  coord_flip() +
  theme(legend.position="none")
```

The following is the output:



*Figure 11.18: Most prevalent emotion in the fetched tweets*

## Conclusion

We started this chapter by discussing how we can fetch data from Twitter. We found that we can fetch data from Twitter for a Query String. The Query String can consist of one word or more. The search can be performed for including all the words or one of the words in the Query String. The query string can also contain hashtag tags combined with regular words.

We then discussed how to fetch data from Twitter for a location and also the data returned by Twitter. We discussed the various uses of analyzing data obtained from Twitter. We studied about a few analyses that we could perform on this data. These analyses consisted of analyzing for countries and places from where the tweets were posted, the language in which the tweets were posted, the devices used to post the tweets, etc. We discussed how to plot the location, from where the tweets were posted, on maps. We also discussed how to create interactive maps.

Lastly, we discussed how to extract the prevalent emotion expressed in a set of tweets fetched from Twitter. In the next and last chapter of this book, we will create a Shiny app using which would provide all the analyses that we discussed in this chapter.

### Points to remember

The rtweet library provides methods for fetching data from Twitter.

The create\_token() function is used to create the Twitter App Token which enables fetching data from Twitter.

The search\_tweets() function can be used for fetching data from Twitter.

The search\_tweets() function returns 90 columns of data.

The lookup\_coords() function returns the geo-coordinates of a location.

The fct\_explicit\_na() function from theforcats library creates an explicit factor for the missing values.

The lat\_lng() function converts the geo-coordinates to the latitude and longitudes.

Using the map() function from the maps library, we can create the map of the world or a specific country.

Using the leaflet() function from the leaflet library, we can create interactive maps.



### Multiple choice questions

The option to set the page length in a data table is:

PageLength

pageLength

pagelength

None of these

The function to fetch tweets using the rtweet package is:

search\_tweet()

search\_tweets()

fetch\_tweet()

fetch\_tweets()

In the function the parameter to provide the geographical location of a particular location is:

geocode

geoCode

locationcode

locationcoords

The escape parameter in the datatable() function:

Exits the program

Stops rendering the data table

Controls whether the HTML code will be parsed or not in the data table

None of these

For using the fct\_explicit\_na() function, which library needs to be included?

forcat

forcats

explicitNA

None of these

## Answers to MCQs

B

B

A

C

B

## Questions

Write a program to only fetch the quoted Tweets and analyze the fetched tweets.

Write a program to stream tweets from a location and analyze the fetched tweets.

## Key terms

**JSON:** JavaScript Object Notation is a lightweight data interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

**URL:** A Uniform Resource Locator, colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

## ***Chidiya***

We have travelled to the final chapter of this book. So far, we discussed about R programming, Shiny programming, and emotion analysis. In [Chapter 11: Emotion Analysis on Twitter Data](#), we discussed the mechanism for extracting emotions expressed in a set of fetched Tweets. In this chapter, we will create a data product (Shiny application) using the program snippets discussed in [Chapter 11: Emotion Analysis on Twitter Data](#). I have named the data product Chidiya. You can use Chidiya in its present form at <https://partha.shinyapps.io/Chidiya>. I may upgrade this in the future.

As we have already discussed most of the programming concepts used in developing Chidiya, I will only dig deep into the new concepts used in developing Chidiya book. For the remaining codes, I will only share the code and state what it does.

## Structure

In this chapter, we will discuss the following topics:

Features of Chidiya

Global code

- Loading the needed libraries
- Global functions
- Initialization code

Server-side code

- Initiating response to user input

Client-side code

The Complete Code for Chidiya

## Objectives

After studying this unit, you should be able to:

Create a data product for emotion analysis using R programming

## Features of Chidiya

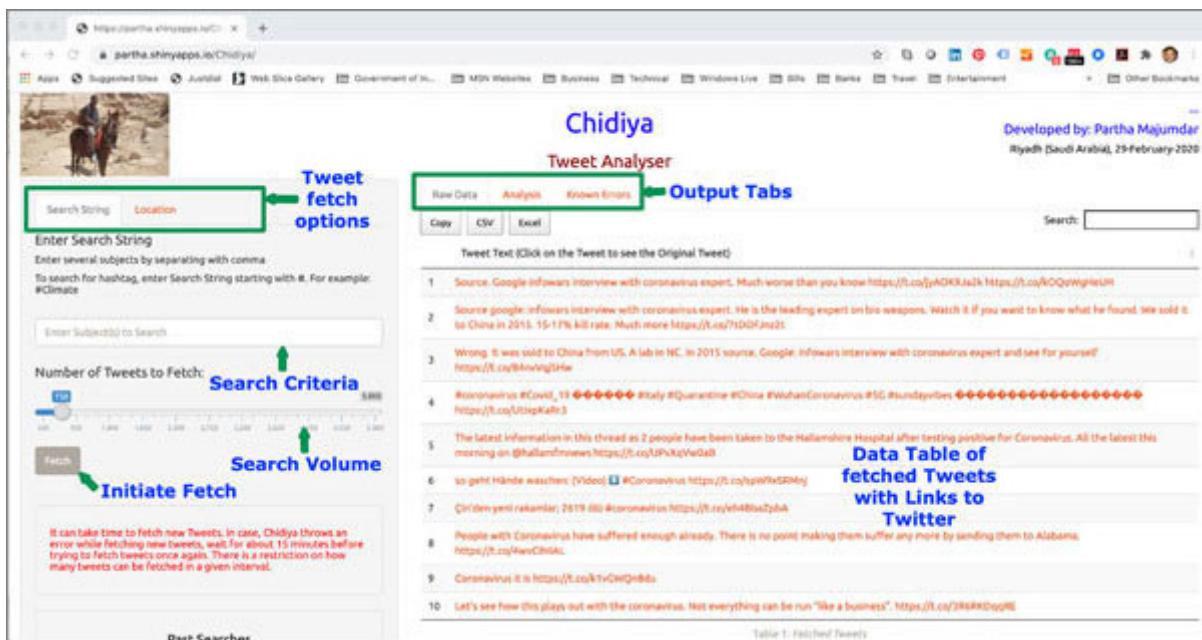
We will start the chapter by discussing the features to develop Chidiya. I will walk you through the GUI. Normally, we would not have the developed GUI at the start of the project. We would develop the prototype using tools like Adobe XD. On getting an approval on the prototype by the sponsors, we would then start the development of the software. However, at this stage, I am discussing a solution that I have already developed and thus, we will take this approach.

**As a practice, I never start writing any code till the prototype is approved by the stakeholders. I prepare the prototype as a paper sketches or using tools like Adobe XD. This saves a lot of rework later in the project. Besides, this is a foundational step in design thinking where the base concept is to involve the customer all through the product development lifecycle.**

## Landing page

Chidiya can be invoked by typing the URL

<https://partha.shinyapps.io/Chidiya> When Chidiya is invoked, the landing page looks as follows:



**Figure 12.1: Chidiya landing page – view 1**

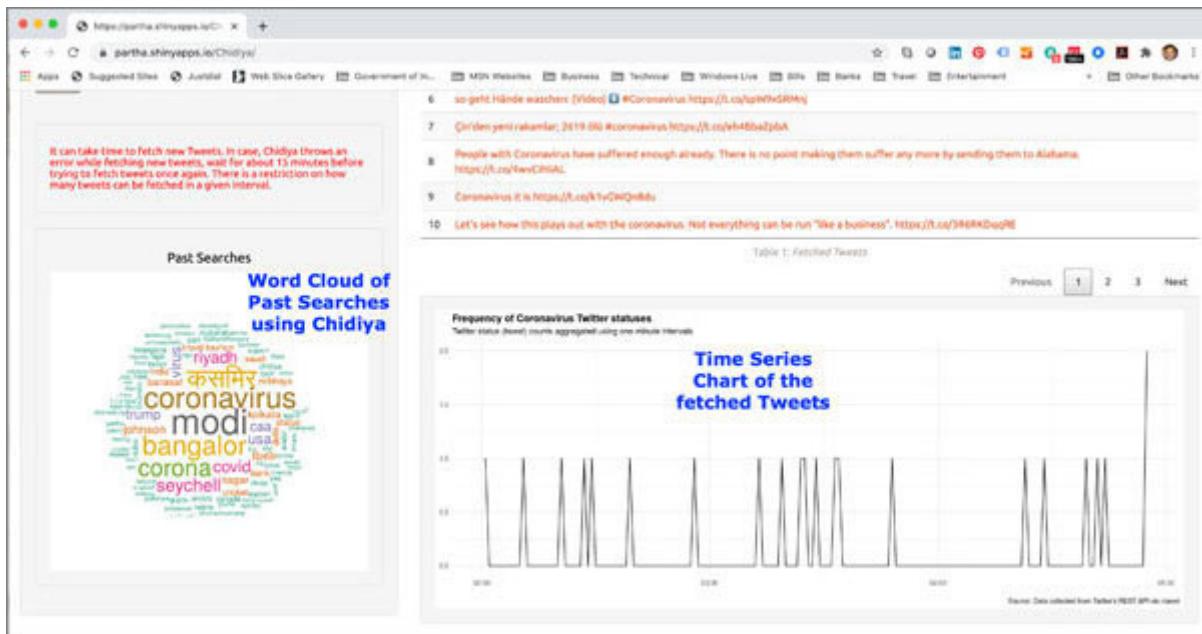
Let us analyze the landing page. The layout of Chidiya is the same as what we discussed in [Chapter 5: Shiny Application](#), [Chapter 6: Shiny Application](#) and [Chapter 9](#): This is so that it is easy to relate to, and we can skip this discussion. However, you can design beautiful UI/UX to use in your application. Shiny applications can support very complex UI/UX.

Below the header on the top, the left-hand panel is the area where the user inputs are accepted. The right-hand panel displays the output. In the left-hand panel, we have a Tabset with two tabs.<sup>2</sup> The default tab is for accepting the search string for which the tweets will be fetched. The user can enter a search string using all the options we discussed in [Chapter 11: Emotion Analysis on Twitter](#). In addition to this, the search string can be provided in any language supported by the computer used for using Chidiya.

We can set the number of tweets to fetch. This is a slider. The default value for the slider is set at 750 tweets. After entering the search string and setting the number of tweets to fetch, the user must click the button titled fetch for Chidiya to start fetching the tweets.

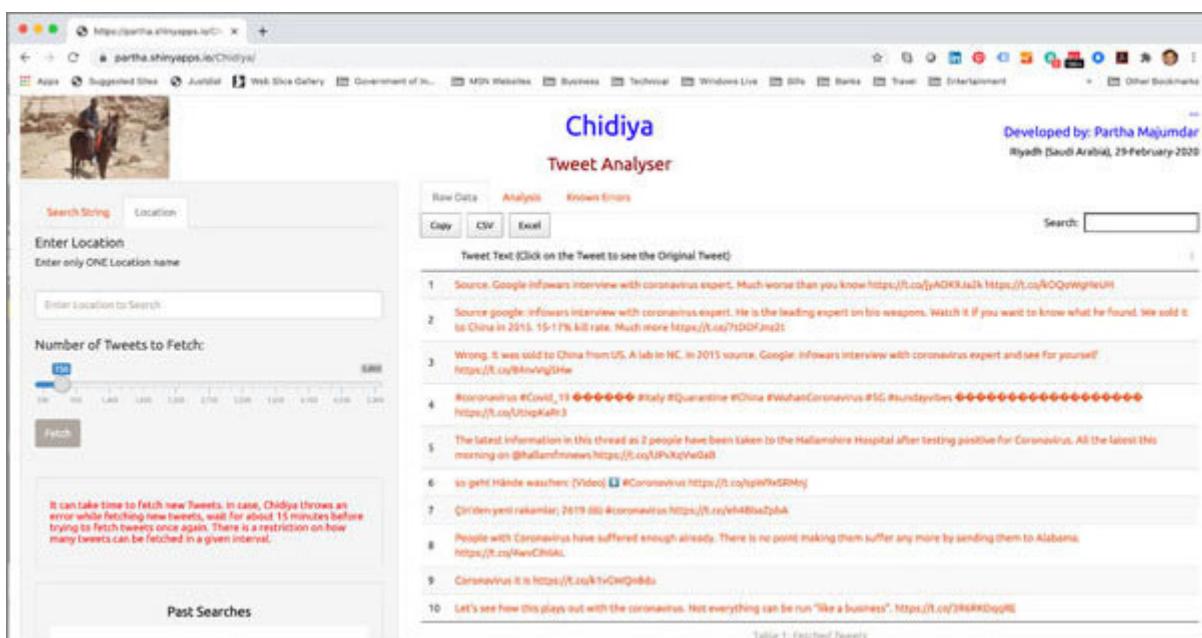
On the right-hand panel, we have a Tabset with three tabs<sup>3</sup> tabs to display the output. The default tab is titled Raw. In this tab, the fetched tweets are listed in a data table. The tweets are presented with a link to the actual tweet in Twitter. So, the user can click on the tweet to view it on Twitter.

When we scroll down this page, we see the following screen. We have a word cloud of the most frequently used Search Strings in the left-hand panel. And in the right-hand panel, we have the time series chart of the fetched tweets:



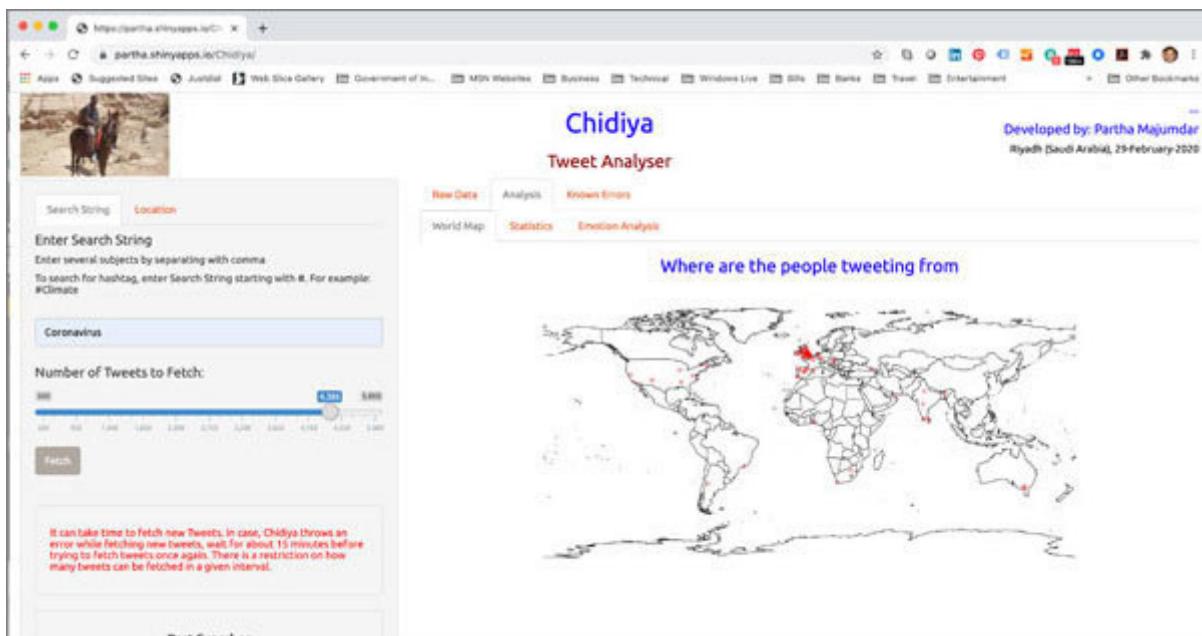
**Figure 12.2:** Chidiya Landing Page – View 2

Next, we will look at the location search facility in the left-hand panel:



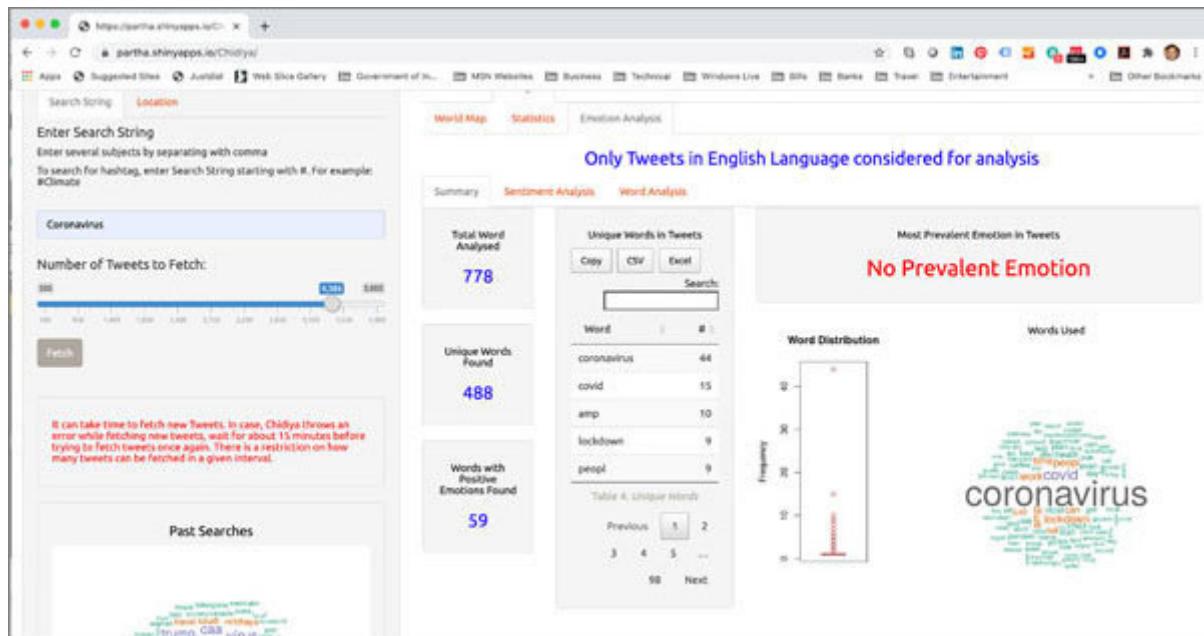
**Figure 12.3:** Location Search facility in the left-hand panel

Here, the user can provide any location pinned on the Google Map. The location can be the name of a country, a city, a street or an exact address. Now, let us turn our attention to the right-hand panel. Under the **Analysis** tab, we have another Tabset shown as follows:



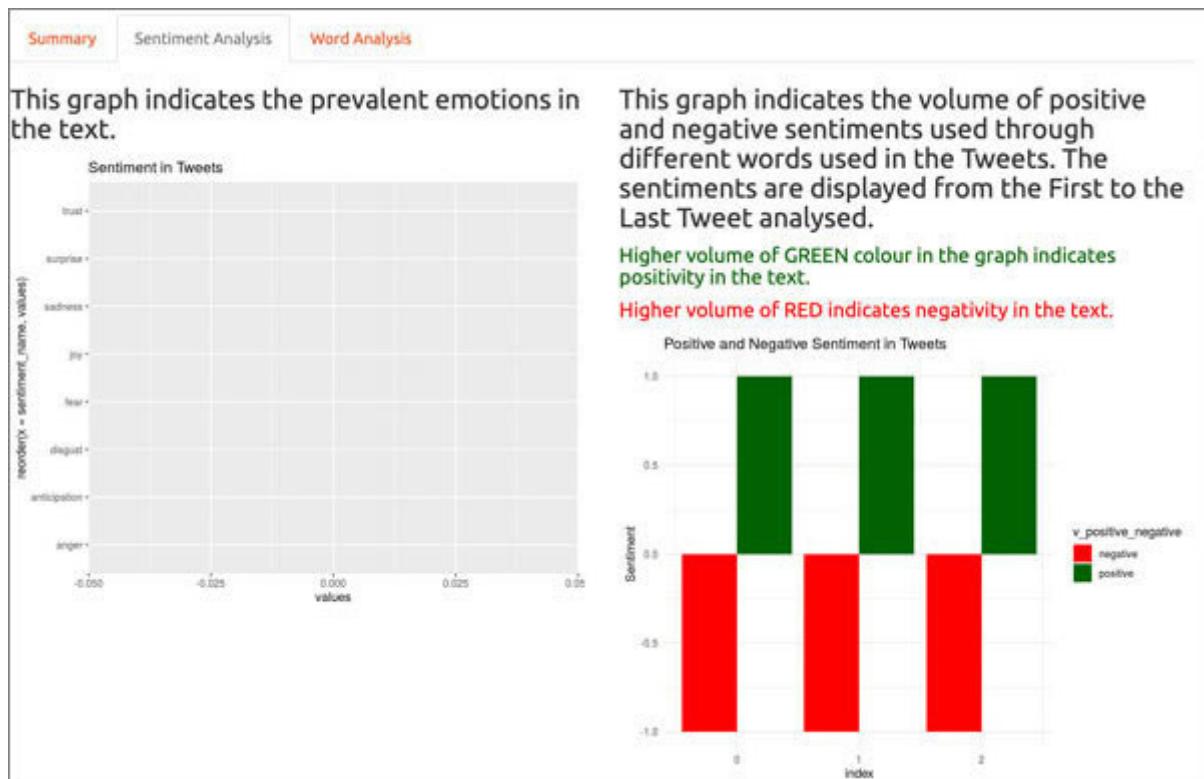
**Figure 12.4:** World Map Display under Analysis

Under the **Analysis** tab, we have three tabs – **World Map**, and **Emotion**. We can see the **World Map** tab in [figure](#). In the following screenshot, we can see the **Emotion Analysis** tab:

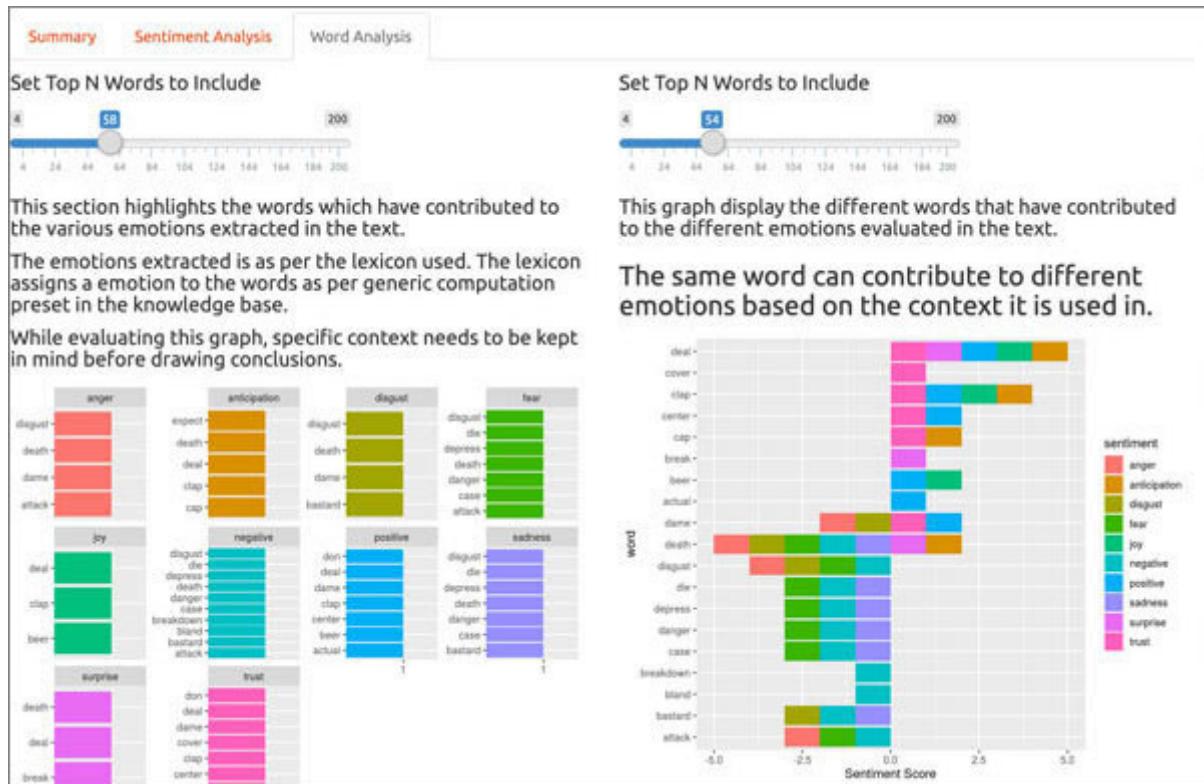


**Figure 12.5:** Summary tab under Emotion Analysis

We see that there are three tabs3 tabs under **Emotion**. In the preceding figure, we can see the **Summary** tab. In the following screenshot, we can see the other two tabs2 tabs under **Emotion**

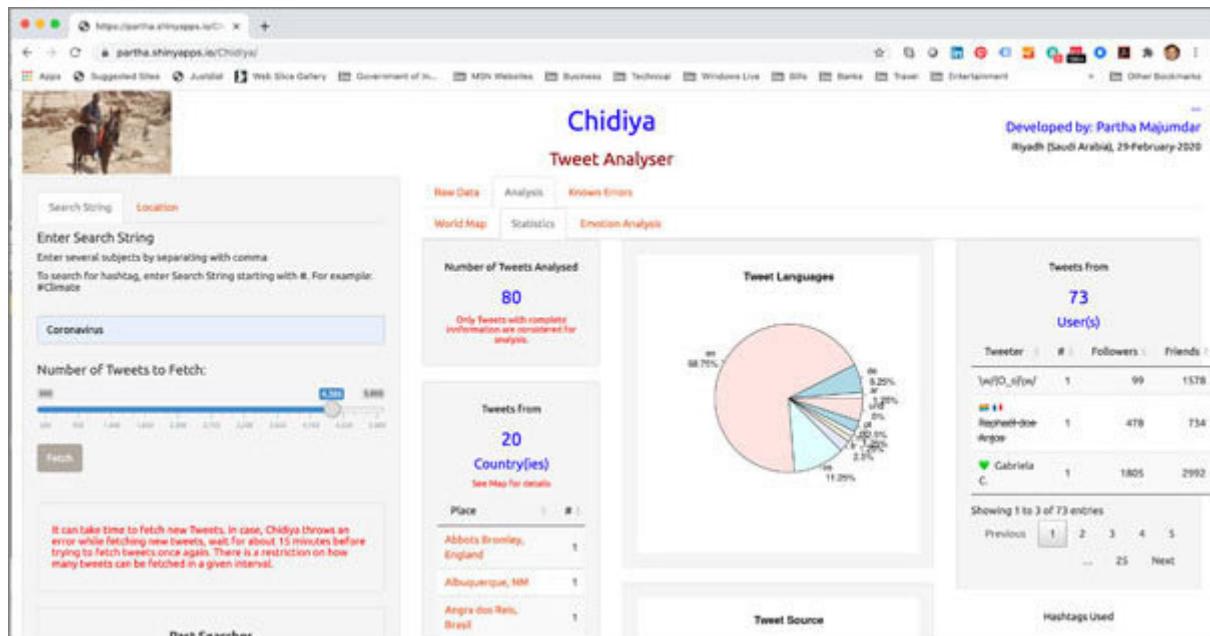


**Figure 12.6: Sentiment Analysis Tab under Emotion Analysis**



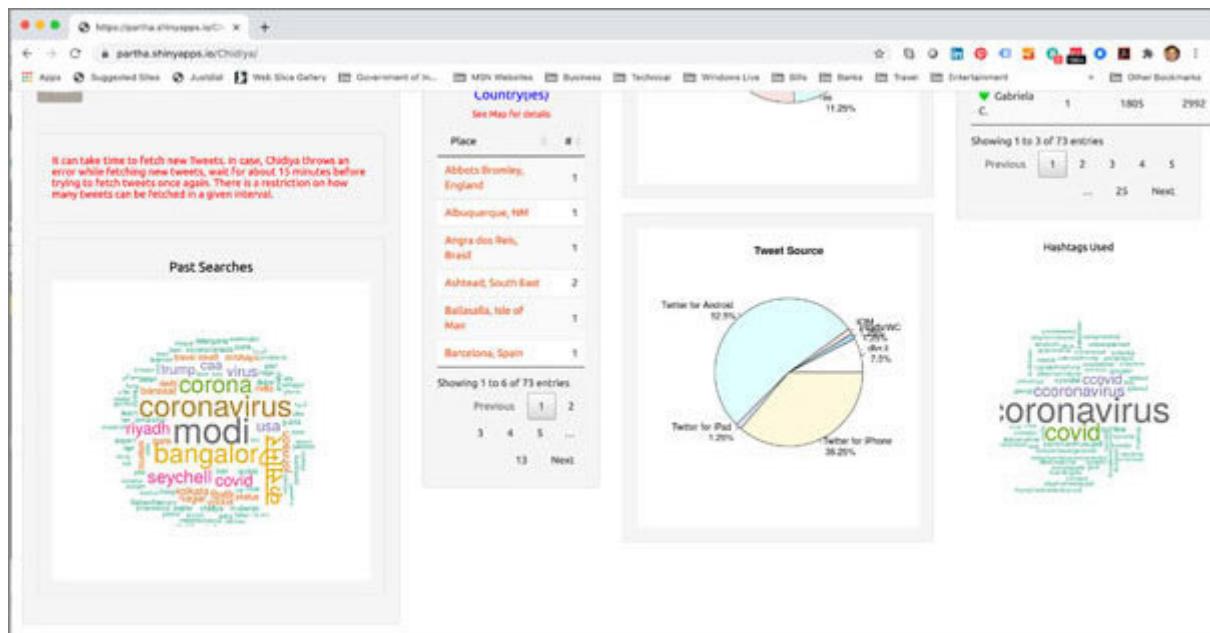
**Figure 12.7: Word Analysis Tab under Emotion Analysis**

The **Statistics** tab under **Emotion Analysis** is shown as follows:



**Figure 12.8: Statistics Tab under Emotion Analysis – View 1**

When we scroll down, we see the following features:



**Figure 12.9: Statistics Tab under Emotion Analysis – View 2**

## Global code

We will start by discussing the code required globally for this application. We declare the libraries globally. Also, we will program some initialization code which can be used by the client-side as well as the server-side code. Lastly, we will globally declare some variables and functions for the program.

**As a general practice, we should make minimum use of global variables. Though, machines have much larger capacities compared to when I used to develop systems in late 1980s and through 1990s, this is a good programming practice and leads to more robust applications.**

## Loading the required libraries

We will use the following code to load the needed libraries:

```
suppressPackageStartupMessages(library(shiny))
suppressPackageStartupMessages(library(shinyjs))
suppressPackageStartupMessages(library(shinythemes))
suppressPackageStartupMessages(library(rtweet))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(ggthemes))
suppressPackageStartupMessages(library(maps))
suppressPackageStartupMessages(library(cli))
suppressPackageStartupMessages(library(gh))
suppressPackageStartupMessages(library(knitr))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(DT))
suppressPackageStartupMessages(library(leaflet))
suppressPackageStartupMessages(library(wordcloud))
suppressPackageStartupMessages(library(rJava))
suppressPackageStartupMessages(library(DBI))
suppressPackageStartupMessages(library(RJDBC))
suppressPackageStartupMessages(library(tm))
suppressPackageStartupMessages(library(SnowballC))
suppressPackageStartupMessages(library(tidyr))
suppressPackageStartupMessages(library(reshape2))
suppressPackageStartupMessages(library(forcats))
```

We will use the `suppressPackageStartupMessages()` function to suppress any messages that are generated when a library is loaded. This can be useful as the messages will get logged on the server. Unnecessary additions to logs can consume space on the server.

Next, we will use the following code to set the HTTP version to be applied for this application. If we do not set the HTTP version, some output from the application may differ when different versions of HTTP are applied by user's machine:

```
httr::set_config(httr::config(http_version = 0))
```

## *Global functions*

We will now discuss the global functions required for Chidiya. Creating these functions makes the code of Chidiya modular. You will notice that we will use considerably less number of code due to the use of these functions.

### Generic function to display a message box

We will create a function to display a message box when the application needs to convey some information and/or instructions to the user. The function creates a modal window.

**In User Interface (UI) design for computer applications, a modal window is a graphical control element subordinate to an application's main window. It creates a mode that disables the main window but keeps it visible, with the modal window as a child window in front of it.**

The code is shown as follows:

```
displayMessage <- function(p_title, p_message) {  
  showModal(modalDialog(  
    title = p_title,  
    p_message,  
    easyClose = TRUE,  
    footer = NULL    ))  
}
```

The `modalDialog()` function is available in the `shiny` library. This function displays a message in a Modal Window. The parameter `title` is optional and sets the Window Title if provided. The parameter `easyClose` takes a Boolean value. If it is set to the window can be dismissed by clicking outside the window.

The showModal() function from the shiny library causes a modal dialog to be displayed in the client browser. Notice that the function displayMessage() takes parameters. The first parameter - p\_title - sets the title for the message box. The second parameter - p\_message - sets the message to be displayed in the message box.

We will also create a special case of the preceding function to show all the error messages shown as follows:

```
displayError <- function(p_error_message) {  
  displayMessage("ERROR", p_error_message)  
}
```

Notice that we have fixed the title of the message box to ERROR in the function

## Function to fetch the subjects of tweets stored in the Chidiya database

We know that Chidiya displays an initial list of tweets. Chidiya fetches these tweets from its database. To restrict the number of initial tweets displayed, Chidiya only displays tweets for one subject. To determine the subject for which the tweets have to be displayed, we need to fetch these subjects and selections. We use the following function to fetch the existing subjects for which tweets are available in the Chidiya database:

```
fetchExistingSubjects <- function() {  
  tryCatch(  
  {  
    v_query <- "SELECT DISTINCT searchString, dateOfEnquiry FROM  
    TB_TWEETS ORDER BY dateOfEnquiry DESC";  
  
    conn = dbConnect(jcc, jdbc_path, user=dsn_uid,  
    password=dsn_pwd)  
    v_rs <- dbSendQuery(conn, v_query);  
    v_existing_subjects <- fetch(v_rs, -1) %>%  
    filter(!is.na(SEARCHSTRING)) %>%  
    filter(!is.na(DATEOFENQUIRY))  
    if(nrow(v_existing_subjects) == 0) {  
      dbDisconnect(conn)  
      displayError("Error in Application Setup. No set up data exists.  
      Quitting")  
      stop(safeError(NULL))  
    }  
  },  
  error = function(e) {  
    dbDisconnect(conn)  
    displayError("Error in Application Setup. An error has occurred.  
    Please contact the administrator.")  
  }  
);  
}  
v_existing_subjects
```

```

dbDisconnect(conn)
},
error = function(e) {
# return a safeError
dbDisconnect(conn)
displayError(paste("Error in Fetching Set Up Data. Quitting", e))
stop(safeError(e))
}
)

return(v_existing_subjects)
}

```

We had discussed the use of the tryCatch() block in [Chapter 9](#): So, it must be clear to you. We will discuss the use of the Database interface as used in this function. We will fetch data from a database in this function. This should be clear from the SQL statement as stated in the variable It is a SELECT statement. I will not discuss SQL in this book and would expect the reader to either have this knowledge or acquire *it* from suitable sources. The programs in this book use basic SQL.

To be able to interact with the database, we must first create a connection with the database. This can be accomplished using the function the RJDBC library. Once the connection to the database has been established, the dbConnect() function returns a connection object. We will store this connection object in the variable We need to use this connection object for every

subsequent interaction with the database till we close the connection.

Using the connection object, we query the database to fetch the previously queried search strings (which is referred to here as subjects) using the function `dbSendQuery()` from the RJDBC library. The function `dbSendQuery()` returns a Result Set which we store in the variable `l`. If the result set does not contain any records, or if we face any other error, we will abort the application. Otherwise, we will process the result set and return the appropriate return value from this function.

In either case of success or failure, we will close the database connection before exiting the function using the `dbDisconnect()` function from the RJDBC library. Notice the use of the `stop()` function. It stops the execution of the current action. The program will quit when it encounters the `stop()` function.

**While using RJDBC, you need being careful about the Java Version installed on your machine. As on date of writing this Book, the Java Version supported by Shiny Applications for JDBC Support is Java Version 1.8.**

You may face issues using the RJDBC library. Please refer to this article for solutions for the same -

### Function to fetch the Previously Searched Subjects

We know that Chidiya displays a word cloud of the search strings that were previously used. Chidiya persists this information in the database. So, we can use the following function to fetch this information. As this information is persisted in the Chidiya database, we need database Interface to get this information as shown in the following code:

```
fetchExistingSearches <- function() {  
  tryCatch(  
    {  
      v_query <- "SELECT Search_String FROM TB_STATISTICS";  
  
      conn = dbConnect(jcc, jdbc_path, user=dsn_uid,  
                      password=dsn_pwd)  
      v_rs <- dbSendQuery(conn, v_query);  
  
      v_existing_searches <- fetch(v_rs, -1) %>%  
        filter(!is.na(SEARCH_STRING))  
      if(nrow(v_existing_searches) == 0) {  
        dbDisconnect(conn)  
        displayError("Error in Application Setup. No set up data (Existing  
                    Searches) exists. Quitting")  
        stop(safeError(NULL))  
      }  
  
      dbDisconnect(conn)
```

```
v_existing_searches <- data.frame(apply(v_existing_searches, 2,
unclass))      },  
  
error = function(e) {  
# return a safeError  
dbDisconnect(conn)  
  
displayError(paste("Error in Fetching Set Up Data. Existing  
Searches do not exist. Quitting", e))  
stop(safeError(e))      }  
)  
  
return(v_existing_searches)  
}
```

After the previous discussion, the contents of this function should be clear.

### Function to read the initial set of tweets

Chidiya displays an initial set of tweets when it is invoked. Chidiya fetches these tweets from its own database and not from Twitter. This is because Chidiya does not decide what subject the user may be interested in at that instance. The code for this is as follows:

```
readTweets <- function(p_search_string, p_search_date) {  
  if(p_search_date == "" || is.null(p_search_date)) {  
    return(NULL)  
  }  
  
  if(p_search_string == "" || is.null(p_search_string)) {  
    return(NULL)  
  }  
  
  v_query = paste("SELECT * FROM TB_TWEETS ",  
  "WHERE searchString = '", p_search_string, "' ",  
  "AND dateOfEnquiry = '", p_search_date, "'", sep = "")  
}  
  
tryCatch(  
{  
  conn = dbConnect(jcc, jdbc_path, user=dsn_uid,  
  password=dsn_pwd)  
  v_rs = dbSendQuery(conn, v_query);
```

```

v_temp = fetch(v_rs, -1);
if(nrow(v_temp) == 0) {
  dbDisconnect(conn)
  displayError("Error in Data Fetch from Database. ZERO ROWS
  Returned. Quitting")

  stop(safeError(NULL))
}

dbDisconnect(conn)
v_temp <- lat_lng(v_temp) %>%
  mutate(tweet_text = TEXT) %>%
  mutate(tweet_language = LANG) %>%
  mutate(tweet_source = SOURCE) %>%
  mutate(tweet_user = NAME) %>%
  mutate(tweet_country = COUNTRY_CODE) %>%
  mutate(tweet_place = PLACE_FULL_NAME) %>%
  mutate(tweet_hashtag = HASHTAGS) %>%
  mutate(tweet_followers_count = FOLLOWERS_COUNT) %>%
  mutate(tweet_friends_count = FRIENDS_COUNT) %>%
  mutate(tweet_urls_expanded_url = STATUS_URL) %>%
  mutate(tweet_place_url = PLACE_URL) %>%
  mutate(tweet_created_at = CREATED_AT) %>%
  mutate(latitude = LAT) %>%
  mutate(longitude = LNG)
},
error = function(e) {
# return a safeError
  dbDisconnect(conn)
  displayError(paste("Error in Data Fetch from Database", e))
}

```

```

stop(safeError(e))
}
)

return(v_temp)
}

```

The preceding code should be clear. However, we will look at one portion of this code given as follows:

```

v_temp <- lat_lng(v_temp) %>%
  mutate(tweet_text = TEXT) %>%
  mutate(tweet_language = LANG) %>%
  mutate(tweet_source = SOURCE) %>%
  mutate(tweet_user = NAME) %>%
  mutate(tweet_country = COUNTRY_CODE) %>%
  mutate(tweet_place = PLACE_FULL_NAME) %>%
  mutate(tweet_hashtag = HASHTAGS) %>%
  mutate(tweet_followers_count = FOLLOWERS_COUNT) %>%
  mutate(tweet_friends_count = FRIENDS_COUNT) %>%
  mutate(tweet_urls_expanded_url = STATUS_URL) %>%
  mutate(tweet_place_url = PLACE_URL) %>%
  mutate(tweet_created_at = CREATED_AT) %>%
  mutate(latitude = LAT) %>%
  mutate(longitude = LNG)

```

Notice that for the columns extracted from the result set, i have created a new column with the prefix This is because the

`dbSendQuery()` function from the RJDBC library returns every column with its name in the upper case. However, the `search_tweets()` function from the rtweet library returns the same columns with the column names in lower case. We will use the return values from both these functions for single use. So, we need to unify the return values from these functions. Thus, I have created a separate set of common columns with the prefix

Instead of creating new columns, I could have renamed the columns as well. However, there are many solutions for the same situation.

## Function to extract words from tweets

We know that we need to extract words from a text for emotion analysis. We have discussed this in [Chapter 8: Emotion Analysis](#) and [Chapter 11: Emotion Analysis on Twitter](#). You will notice that in [Chapter 11: Emotion Analysis on Twitter](#) we had used the same code shown as follows. So, I will present the code and not discuss it further.

Notice that, to this function, we pass the data frame containing the tweets as a parameter. Also, notice that we do not quit Chidiya if we encounter an error in this function:

```
extractWordsFromTweets <- function(p_tweet_data_frame) {  
  tryCatch(  
  {  
    # Remove the Emoticons  
    p_tweet_data_frame <- gsub("[^\x01-\x7F]", "", p_tweet_data_frame)  
  
    # Form the Corpus  
    v_corpus <- VCorpus(VectorSource(p_tweet_data_frame))  
  
    # Remove URLs  
    removeURL <- content_transformer(function(x) gsub("(    toSpace <- content_transformer(function (x, pattern) gsub(pattern, " ", x))
```

```

v_corpus <- tm_map(v_corpus, removeURL)
v_corpus <- tm_map(v_corpus, toSpace, "/")
v_corpus <- tm_map(v_corpus, toSpace, "@")
v_corpus <- tm_map(v_corpus, toSpace, "\\|")

v_corpus <- tm_map(v_corpus, toSpace, ''))
v_corpus <- tm_map(v_corpus, toSpace, '')

# Create the Term Document Matrix after cleaning the Corpus
v_tdm <- TermDocumentMatrix(v_corpus,
  control =
list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf)))
)
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))
v_data <- subset(v_data, !(ind == "" || is.null(ind) || is.na(ind)))
},
error = function(e) {

```

```
displayError(paste("Error in Extracting Words from Tweets. This  
could be because the search does not produce any words in  
English Language. Please Try Again.", e, sep = " "))  
}  
)  
  
return(v_data)  
}
```

## Function to extract emotions from tweets

We have discussed the code for this function in [Chapter 11: Emotion Analysis on Twitter](#). Take a look at the following code:

```
determineEmotionCount <- function(p_tidy_data) {  
  v_combined <- sort(union(levels(p_tidy_data$word),  
    levels(v_nrc$word)))  
  v_text_sentiment2 <- inner_join(mutate(p_tidy_data,  
    word=factor(word, levels=v_combined)),  
    mutate(v_nrc, word=factor(word, levels=v_combined)))  
}  
v_text_sentiment1 <- v_text_sentiment2 %>%  
  count(word, index = row_number() %/% 80, sentiment) %>%  
  mutate(v_original_n = n) %>%  
  mutate(v_positive_negative = ifelse(sentiment %in%  
    v_positive_emotions,  
    'positive', 'negative')) %>%  
  mutate(v_original_sentiment = sentiment) %>%  
  spread(sentiment, n, fill = 0)  
v_text_sentiment <- v_text_sentiment1 %>%  
  mutate(positive = ifelse("positive" %in%  
    colnames(v_text_sentiment1), positive, 0)) %>%  
  mutate(joy = ifelse("joy" %in% colnames(v_text_sentiment1), joy,  
    0)) %>%  
  mutate(surprise = ifelse("surprise" %in%  
    colnames(v_text_sentiment1), surprise, 0)) %>%
```

```
mutate(trust = ifelse("trust" %in% colnames(v_text_sentiment1),
trust, 0)) %>%
mutate(anticipation = ifelse("anticipation" %in%
colnames(v_text_sentiment1), anticipation, 0)) %>%
mutate(negative = ifelse("negative" %in%
colnames(v_text_sentiment1), negative, 0)) %>%
mutate(anger = ifelse("anger" %in% colnames(v_text_sentiment1),
anger, 0)) %>%
mutate(disgust = ifelse("disgust" %in%
colnames(v_text_sentiment1), disgust, 0)) %>%
mutate(sadness = ifelse("sadness" %in%
colnames(v_text_sentiment1), sadness, 0)) %>%
mutate(fear = ifelse("fear" %in% colnames(v_text_sentiment1), fear,
0)) %>%
mutate(sentiment = (positive+joy+surprise+trust+anticipation) -
(negative+anger+disgust+sadness+fear)) %>%
ungroup()
return(v_text_sentiment)
}
```

### Function to save statistics

The last function we need is to save the statistics of the usage of Chidiya to the database. The code is shown as follows:

```
saveStatistics <- function() {  
  tryCatch(  
  {  
    v_statement <- paste("INSERT INTO TB_STATISTICS (",  
    "Date_of_Search,",  
    "Search_String,",  
    "No_requested,",  
    "No_of_tweets_fetched,",  
    "No_of_tweets_analysed,",  
    ",",  
    "No_of_hashtag_found,",  
    "No_of_countries,",  
    "No_of_places,",  
    "No_of_languages,",  
    "No_of_devices,",  
    "No_of_users,",  
    ",",  
    "Total_words,",  
    "Unique_words,",  
    "Prevalent_emotion)",  
    "VALUES (",  
    paste("", v_selected_date, '", sep = ""),  
    paste("", v_selected_subject, '", sep = ""),
```

```

paste(v_number_requested, "", sep = ""),
paste(v_number_fetched, "", sep = ""),
paste(v_number_analysed, "", sep = ""),

paste(v_number_hashtag_found, "", sep = ""),
paste(v_number_countries, "", sep = ""),
paste(v_number_places, "", sep = ""),
paste(v_number_languages, "", sep = ""),
paste(v_number_devices, "", sep = ""),
paste(v_number_users, "", sep = ""),
paste(v_number_words, "", sep = ""),
paste(v_number_unique_words, "", sep = ""),
paste("", v_prevalent_emotion, "", sep=""),
")", sep = " ")
conn = dbConnect(jcc, jdbc_path, user=dsn_uid,
password=dsn_pwd)
if(!dbSendUpdate(conn, v_statement)) {
displayError("Error Writing to Database.")
} else {
dbCommit(conn)
}
dbDisconnect(conn)
},
error = function(e) {
dbDisconnect(conn)
}}}
```

In this function, we will form an INSERT statement and use the dbSendUpdate() function from the RJDBC library to update the table in the database. If the INSERT statement executes appropriately, we will use the dbCommit() function from the RJDBC library to commit the transaction.



### *Initialization code*

We will go through the code we need for initializing the application. One part of initialization process is to load the libraries; we have discussed that already. We will go through the rest of the code.

## Reading the system parameters

The first thing we need to do is to read the system parameters.  
We have discussed this in [Chapter 11: Emotion Analysis on Twitter](#)

```
##### READ THE SYSTEM PARAMETERS
#####
v_system_parameters <- read.csv(file = "SystemParameters.csv",
header = TRUE)
attach(v_system_parameters)
```

## Setting up the Twitter Account

Next, we will set up a Twitter account using the following code. We have discussed this in [Chapter 11: Emotion Analysis on Twitter](#)

```
##### SET UP THE TWITTER ACCOUNT
#####
api_key <- as.character(v_system_parameters[(which(Parameter ==
"api_key")),]$Value)
api_secret_key <-
as.character(v_system_parameters[(which(Parameter ==
"api_secret_key")),]$Value)
access_token <- as.character(v_system_parameters[(which(Parameter ==
"access_token")),]$Value)
access_token_secret <-
as.character(v_system_parameters[(which(Parameter ==
"access_token_secret")),]$Value)

## Authenticate via Web Browser
token <- create_token(
app = "ParthaEmotions",
consumer_key = api_key,
consumer_secret = api_secret_key,
access_token = access_token,
access_secret = access_token_secret)
```

## Setting up the database

Next, we will set up the database using the following code:

```
##### SET UP THE DATABASE
#####
dsn_driver = as.character(v_system_parameters[(which(Parameter ==
"dsn_driver")),]$Value)
dsn_database = as.character(v_system_parameters[(which(Parameter ==
"dsn_database")),]$Value)
dsn_hostname = as.character(v_system_parameters[(which(Parameter ==
"dsn_hostname")),]$Value)
dsn_port = as.character(v_system_parameters[(which(Parameter ==
"dsn_port")),]$Value)
dsn_protocol = as.character(v_system_parameters[(which(Parameter ==
"dsn_protocol")),]$Value)
dsn_uid = as.character(v_system_parameters[(which(Parameter ==
"DBUserID")),]$Value)
dsn_pwd = as.character(v_system_parameters[(which(Parameter ==
"DBPassword")),]$Value)
## Connect to the Database
jcc = JDBC(dsn_driver, "Drivers/db2jcc4.jar", identifier.quote="\"");
jdbc_path = paste("jdbc:db2://", dsn_hostname, ":", dsn_port, "/",
dsn_database, sep="");
```

The important thing to notice here is that we must install the driver for the database in our machine. For Chidiya, I have used the DB2 database and the driver file is The appropriate path for

the database driver file needs to be provided to the JDBC() function from the RJDBC library.

While publishing the Shiny application to the shinyappsi server, it is important to ensure that the database drivers are present in A safer option is to store the driver files driver files in the same directory where the Shiny application files are stored. And then include the driver files in the Shiny application files while publishing it application to We have discussed how to publish a Shiny application to shinyappsi in [Chapter 4: Structure of Shiny](#)

## Setting up the Google Maps API

Next, we will set up the Google Maps API using the following code:

```
##### SET UP THE GOOGLE MAPS
#####
google_maps_api_key =
as.character(v_system_parameters[(which(Parameter ==
"google_maps_api_key")),$Value])
```

## Setting up the Lexicon

Next, we will set up the lexicon using the following code. We know that we have to use the nrc lexicon:

```
##### READ THE LEXICON
#####
v_nrc <- get_sentiments("nrc")
v_positive_emotions <- c("positive","joy","anticipation","surprise",
"trust")
v_negative_emotions <- c("negative","anger","disgust","fear",
"sadness")
v_positive_sentiment <- v_nrc %>%
filter(sentiment %in% v_positive_emotions)
```

## Setting up the data for initial display

Lastly, we will do the needful for setting up the initial display. The following code should be straightforward to understand:

```
##### SET UP INITIAL DATA TO DISPLAY
#####
# Fetch existing searches
v_existing_searches <- fetchExistingSearches()

# Fetch the Initial Set of Tweets from Database to Display
v_existing_subjects <- fetchExistingSubjects()

v_selected_subject <- v_existing_subjects[1]$SEARCHSTRING
v_selected_date <- v_existing_subjects[1]$DATEOFENQUIRY

v_tweets_to_display <- readTweets(v_selected_subject,
v_selected_date)

v_number_requested <- as.integer(o)
v_number_fetched <- as.integer(o)
v_number_analysed <- as.integer(o)
v_number_hashtag_found <- as.integer(o)
v_number_countries <- as.integer(o)
v_number_places <- as.integer(o)
v_number_languages <- as.integer(o)
v_number_devices <- as.integer(o)
```

```
v_number_users <- as.integer(0)
v_number_words <- as.integer(0)
v_number_unique_words <- as.integer(0)
v_prevalent_emotion <- ""
```

Notice that we have started making use of the functions that we declared. Take a note of the two variables declared in the preceding figure – v\_selected\_subject and I will use these variables throughout the program to store the current subject (search string or location) for which the tweets are being analyzed and the current set of tweets that are being analyzed respectively.

The set of variables with prefix v\_number\_ and v\_prevalent\_emotion is used to store the statistics that we will write to the Chidiya database, every time Chidiya is used. Lastly, we will set up the initial emotion analysis requisites with the following code:

```
# Set up initial Emotion Analysis
tryCatch(
{
  v_words_in_tweets <- extractWordsFromTweets(
    (v_tweets_to_display %>%
      filter(tweet_language == 'en'))$tweet_text
  ) %>%
    rename(word = ind) %>%
    rename(n = values)
},
error = function(e) {
  # return a safeError
```

```
displayError(paste("Error Setting up Chidiya. Quitting", e, sep ="
"))
stop(safeError(e))
}
)
v_combined <- sort(union(
levels(v_words_in_tweets$word),
levels(v_positive_sentiment$word)))
v_positive_sentiment_words <-
semi_join(mutate(v_words_in_tweets,
word=factor(word, levels=v_combined)),
mutate(v_positive_sentiment,
word=factor(word, levels=v_combined))) %>%
count(word, sort = TRUE)
```

Note that the union() function is available in many libraries in R for different purposes. I have used the union() function from the dplyr library to make a union between the two sets in the preceding code.

## Server-side code

Now, we will discuss the server-side code of Chidiya. I will discuss the server-side code before the client-side code as I feel that the dependence on the server-side code is more for the client-side code than the other way around.

I will only discuss the code which introduces new techniques and concepts in this chapter. The complete code for Chidiya has been provided as follows. I would expect that you will be able to understand the code, which uses the techniques and concepts already discussed in the book so far, on your own:

## Initiating response to user input

We will take two sets of user inputs in Chidiya. When the users search for tweets for a search string, we will accept the search string and the number of tweets to fetch. Similarly, when the user searches for tweets for a location, we will accept the name of the location and the number of tweets to fetch. We know from [Chapter 6: Shiny Application 2](#) that we need two sets of controls for this purpose. We will do the same in Chidiya as well.

In Chidiya, once these inputs have been provided, the user must click the **Fetch** button for Chidiya to initiate the process for fetching the tweets. So, we have two **Fetch** buttons as well – one for searching by search string and one for searching for a location. In the client-side code, I have named the **Fetch** button for searching by search string as And I have named the **Fetch** button for searching for a location as

The following code responds to the user when the **Fetch** button is clicked for searching for tweets based on a search string:

```
observeEvent(input$v_fetch,
{
  if(!(input$v_search_string == "" || is.null(input$v_search_string))) {
    tryCatch(
    {
      withProgress(message = 'Fetching Tweets for Search String',
```

```

detail = 'This may take a while...', value = 100, {
v_tweets_to_display <<- v_tweets(input$v_search_string,
input$v_number_of_tweets)
extractWords()

})

refreshDisplay()
# Save Statistics
saveStatistics()
},
error = function(e) {
displayError(paste("Error Fetching Tweets. This could be because
the search does not fetch any Tweet OR the fetched Tweets does
not contain any Words in English Language OR the search was
for too few numbers of Tweets. The Graphs and other Outputs
will not be updated due to this. Try fetching larger number of
Tweets OR using Search Terms in English Language.", e, sep =
})
}

# Reset Display to default TAB
updateTabsetPanel(session, "main_panel",
selected = "Raw Data"
)
}
}
)

```

The core of the preceding code is in the `tryCatch()` block. So, I will rewrite the same:

```
tryCatch(
```

```

{
withProgress(message = 'Fetching Tweets for Search String',

detail = 'This may take a while...', value = 100, {
v_tweets_to_display <<- v_tweets(input$v_search_string,
input$v_number_of_tweets)
extractWords()
})
refreshDisplay()
# Save Statistics
saveStatistics()
},
error = function(e) {
displayError(paste("Error Fetching Tweets. This could be because
the search does not fetch any Tweet OR the fetched Tweets does
not contain any Words in English Language OR the search was
for too few numbers of Tweets. The Graphs and other Outputs
will not be updated due to this. Try fetching larger number of
Tweets OR using Search Terms in English Language.", e, sep =
"))
}
)
}
```

We see that we call four functions when the **Fetch** button is clicked. They are listed as follows:

**v\_tweets:** This function fetches the tweets. This function is specific for fetching tweets for a search string. Notice that the output of this function is captured in a global variable

**extractWords:** This function is common for when tweets are fetched based on a search string or when tweets are fetched for a location. This function extracts the words in the fetched tweets.

**refreshDisplay:** This function refreshes the complete output from Chidiya. We will discuss this function further. This function makes Chidiya responsive to the user inputs.

**saveStatistics:** This function saves the details of the Chidiya usage for this instance by the user. We have discussed this function earlier in this chapter.

Now, let us look at the code for responding to user input when the user wants to fetch tweets for a location. The code is as shown below:

```
observeEvent(input$v_fetch_location,
{
  if(!(input$v_search_location == "" ||
  is.null(input$v_search_location))) {
    tryCatch(
    {
      withProgress(message = 'Fetching Tweets for Location',
      detail = 'This may take a while...', value = 100, {
        v_tweets_to_display <-
        v_tweets_for_location(input$v_search_location,
        input$v_number_of_tweets_location)
        extractWords()
      })
      refreshDisplay()
    })
  }
})
```

```

# Save Statistics
saveStatistics()
},

error = function(e) {
displayError(paste("Error Fetching Tweets. This could be because
the search does not fetch any Tweet OR the fetched Tweets does
not contain any Words in English Language OR the search was
for too few numbers of Tweets. The Graphs and other Outputs
will not be updated due to this. Try fetching larger number of
Tweets OR using Search Terms in English Language.", e, sep =
"))
}
)
}

# Reset Display to default TAB
updateTabsetPanel(session, "main_panel",
selected = "Raw Data"
)
}
}
)

```

Notice that this code is the same except for the call to the v\_tweets\_for\_location() function.

## Fetching the tweets

We know from [Chapter 11: Emotion Analysis on Twitter Data](#) that we have to use different calls to the `search_tweets()` function for searching tweets based on search string and location. So, we will have to use different functions for this purpose.

The code for fetching tweets based on search string is provided as follows:

```
v_tweets <- function(p_search_string, p_number_of_tweets) {  
  tryCatch(  
  {  
    shinyjs::hide("OutputTabsEmotionAnalysis")  
    shinyjs::show("OutputTabsNoEmotionAnalysis")  
    shinyjs::hide("statisticsTab")  
    shinyjs::show("NoStatistics")  
    shinyjs::hide("worldMapDisplay")  
    shinyjs::show("NoWorldMap")  
  
    v_temp <- search_tweets(  
      p_search_string,  
      '-filter' = "replies",  
      n = p_number_of_tweets,  
      include_rts = FALSE,  
      retryonratelimit = FALSE #,  
    )
```

```
v_temp <- gatherTweetData(v_temp, p_search_string,  
p_number_of_tweets)
```

```
shinyjs::show("OutputTabsEmotionAnalysis")  
shinyjs::hide("OutputTabsNoEmotionAnalysis")
```

```
shinyjs::show("statisticsTab")  
shinyjs::hide("NoStatistics")  
shinyjs::show("worldMapDisplay")  
shinyjs::hide("NoWorldMap")  
},  
error = function(e) {  
  displayError(paste("Error Fetching Tweets.", e))  
}  
)
```

```
return(v_temp)  
}
```

And the code for fetching tweets for a location is shown as follows:

```
v_tweets_for_location <- function(p_search_location,  
p_number_of_tweets) {  
tryCatch(  
{  
  shinyjs::hide("OutputTabsEmotionAnalysis")  
  shinyjs::show("OutputTabsNoEmotionAnalysis")  
  shinyjs::hide("statisticsTab")
```

```
shinyjs::show("NoStatistics")
shinyjs::hide("worldMapDisplay")
shinyjs::show("NoWorldMap")

v_temp <- search_tweets(
  geocode = lookup_coords(p_search_location, apikey =
    google_maps_api_key),
  '-filter' = "replies",

  n = p_number_of_tweets,
  include_rts = FALSE,
  retryonratelimit = FALSE #,
)

v_temp <- gatherTweetData(v_temp, p_search_location,
  p_number_of_tweets)

shinyjs::show("OutputTabsEmotionAnalysis")
shinyjs::hide("OutputTabsNoEmotionAnalysis")
shinyjs::show("statisticsTab")
shinyjs::hide("NoStatistics")
shinyjs::show("worldMapDisplay")
shinyjs::hide("NoWorldMap")
},
error = function(e) {
  displayError(paste("Error Fetching Tweets.", e))
}
)

return(v_temp)
}
```

There is only one line of code which is different between the two functions. Can you think of a better way to write this code?

## Making Chidiya responsive

The function refreshDisplay() is used to update all the output elements of Chidiya when a new user input is provided. The code is shown in the following code:

```
refreshDisplay <- function() {  
  # Dummy  
  determineLanguageCount(FALSE)  
  determineSourceCount(FALSE)  
  determineUserCount()  
  determineCountryCount()  
  determinePlaceCount(FALSE)  
  showUniqueWords(FALSE)  
  showMostPrevalentEmotion()  
  
  # Render on Fetch  
  output$tweets <- renderDataTable({showTweets()})  
  output$tweetTimeSeries <- renderPlot({showTweetTimeSeries()})  
  output$WorldMap <- renderPlot({showWorldMap()})  
  output$numberOfTweets <- renderText({nrow(v_tweets_to_display)})  
  output$languageCount <- renderPlot({determineLanguageCount()})  
  output$sourceCount <- renderPlot({determineSourceCount()})  
  output$userCount <- renderText({determineUserCount()})  
  output$countryCount <- renderText({determineCountryCount()})  
  
  output$placeCount <- renderDataTable({determinePlaceCount()})
```

```

output$mostTweeter <- renderDataTable({determineMostTweeter()})
output$hashTagWordCloud <-
renderPlot({showHashTagWordCloud()})

output$totalWordCount <-
renderText({return(sum(v_words_in_tweets$n))})
output$uniqueWordCount <-
renderText({return(nrow(v_words_in_tweets))})
output$positiveSentimentWordCount <-
renderText({return(nrow(v_positive_sentiment_words))})
output$uniqueWords <- renderDataTable({showUniqueWords()})
output$boxPlot <- renderPlot({showBoxPlot()})
output$wordWordCloud <- renderPlot({showWordWordCloud()})
output$wordContributionToSentiments <-
renderPlot({showWordContributionToSentiments()})
output$sentiments <- renderPlot({showSentiments()})
output$positiveNegativeSentiments <-
renderPlot({showPositiveNegativeSentiments()})
output$mostPrevalentEmotion <-
renderText({showMostPrevalentEmotion()})
output$sentimentScore <- renderPlot({determineSentimentScore()})
}

```

Notice that we update every Output variable in this function. There is a pattern in this code. We will discuss this pattern through one example and then the rest of the code should be possible for you to understand on your own.

Let us study this code in detail:

```
output$tweets <- renderDataTable({showTweets()})
```

We notice that we call the function `showTweets()` in the `render` function. The code for the `showTweets()` function is shown in the following code:

```
showTweets <- function() {  
  withProgress(message = 'Tabulating Tweets',  
    detail = 'This may take a while...', value = 100, {  
      v_temp <- v_tweets_to_display %>%  
        mutate(tweet_with_link = paste("href=\"",  
          tweet_urls_expanded_url,  
          "", target='_blank'>",  
          tweet_text,  
          "",  
          sep = "")  
    })  
    DT::datatable(  
      {select(v_temp, tweet_with_link)},  
      colnames = c('Tweet Text (Click on the Tweet to see the Original  
Tweet)'),  
      caption = htmltools::tags$caption(  
        style = 'caption-side: bottom; text-align: center;',  
        'Table 1: ',  
        htmltools::em('Fetched Tweets')  
      ),  
      extensions = 'Buttons',  
  
      escape = FALSE,  
      options = list(  
        fixedColumns = TRUE,
```

```
autoWidth = FALSE,  
ordering = TRUE,  
pageLength = 10,  
dom = 'Bftsp',  
buttons = c('copy', 'csv', 'excel')  
))  
}  
}
```

This code should be clear as we have discussed it in [Chapter 11: Emotion Analysis on Twitter](#). The rest of the server-side code of Chidiya follows the same pattern and so, should be easy to follow. The complete code is provided in this chapter.

### **Client-side code**

There is no new aspect or concept in the client-side code of Chidiya that we have not discussed in this book already. So, I will not go through this code piece-by-piece. I have provided the complete code for Chidiya and I would expect that you understand the client-side code from the same.

## The complete for code Chidiya

The complete code for Chidiya is provided below. The code is also available at I have purposely not stated this in the preceding codes and instructions that I have not used the reactive() function at all in the code for Chidiya, though Chidiya is a fully responsive application. This is a major differentiation with the code that we discussed in [Chapter 9:](#)

```
##### LOAD THE REQUIRED PACKAGES
#####
suppressPackageStartupMessages(library(shiny))
suppressPackageStartupMessages(library(shinyjs))
suppressPackageStartupMessages(library(shinythemes))
suppressPackageStartupMessages(library(rtweet))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(ggthemes))
suppressPackageStartupMessages(library(maps))
suppressPackageStartupMessages(library(cli))
suppressPackageStartupMessages(library(gh))
suppressPackageStartupMessages(library(knitr))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(DT))
suppressPackageStartupMessages(library(leaflet))
suppressPackageStartupMessages(library(wordcloud))
suppressPackageStartupMessages(library(rJava))
suppressPackageStartupMessages(library(DBI))
suppressPackageStartupMessages(library(RJDBC))
```

```
suppressPackageStartupMessages(library(tm))
suppressPackageStartupMessages(library(SnowballC))
```

```
suppressPackageStartupMessages(library(tidyr))
suppressPackageStartupMessages(library(reshape2))
suppressPackageStartupMessages(library(forcats))
```

```
httr::set_config(httr::config(http_version = 0))
```

```
##### REQUIRED FUNCTIONS
```

```
#####
```

```
## DISPLAY A MESSAGE BOX
```

```
displayMessage <- function(p_title, p_message) {
  showModal(modalDialog(
    title = p_title,
    p_message,
    easyClose = TRUE,
    footer = NULL
  ))
}
```

```
## DISPLAY AN ERROR MESSAGE BOX
```

```
displayError <- function(p_error_message) {
  displayMessage("ERROR", p_error_message)
}
```

```
## FETCH THE EXISTING SUBJECTS FOR WHICH TWEETS EXIST
IN THE DATABASE
```

```
fetchExistingSubjects <- function() {
  tryCatch(
{
```

```
v_query <- "SELECT DISTINCT searchString, dateOfEnquiry FROM  
TB_TWEETS ORDER BY dateOfEnquiry DESC";
```

```
conn = dbConnect(jcc, jdbc_path, user=dsn_uid,  
password=dsn_pwd)  
v_rs <- dbSendQuery(conn, v_query);
```

```
v_existing_subjects <- fetch(v_rs, -1) %>%  
filter(!is.na(SEARCHSTRING)) %>%  
filter(!is.na(DATEOFENQUIRY))  
if(nrow(v_existing_subjects) == 0) {  
dbDisconnect(conn)  
displayError("Error in Application set-up. No set-up data exists.  
Quitting")  
stop(safeError(NULL))  
}
```

```
dbDisconnect(conn)  
},  
error = function(e) {  
# return a safeError  
dbDisconnect(conn)  
displayError(paste("Error in Fetching Set Up Data. Quitting", e))  
stop(safeError(e))  
}  
)
```

```
return(v_existing_subjects)  
}
```

```
## FETCH THE SUBJECTS FOR WHICH TWEETS WERE
SEARCHED IN THE PAST
fetchExistingSearches <- function() {
tryCatch(
  {

v_query <- "SELECT Search_String FROM TB_STATISTICS";

conn = dbConnect(jcc, jdbc_path, user=dsn_uid,
password=dsn_pwd)
v_rs <- dbSendQuery(conn, v_query);

v_existing_searches <- fetch(v_rs, -1) %>%
filter(!is.na(SEARCH_STRING))
if(nrow(v_existing_searches) == 0) {
  dbDisconnect(conn)
  displayError("Error in Application Setup. No set up data (Existing
Searches) exists. Quitting")
  stop(safeError(NULL))
}

dbDisconnect(conn)

v_existing_searches <- data.frame(apply(v_existing_searches, 2,
unclass))
},
error = function(e) {
# return a safeError
dbDisconnect(conn)
```

```
displayError(paste("Error in Fetching Set Up Data. Existing
Searches do not exist. Quitting", e))
stop(safeError(e))
}

)

return(v_existing_searches)

}

## READ TWEETS FROM THE DATABASE FOR A PARTICULAR
SEARCH CRITERION
readTweets <- function(p_search_string, p_search_date) {
if(p_search_date == "" || is.null(p_search_date)) {
return(NULL)
}

if(p_search_string == "" || is.null(p_search_string)) {
return(NULL)
}

v_query = paste("SELECT * FROM TB_TWEETS ",
"WHERE searchString = '", p_search_string, "' ",
"AND dateOfEnquiry = '", p_search_date, "'",
sep = ""
)

tryCatch(
{
```

```
conn = dbConnect(jcc, jdbc_path, user=dsn_uid,
password=dsn_pwd)
v_rs = dbSendQuery(conn, v_query);

v_temp = fetch(v_rs, -1);
if(nrow(v_temp) == 0) {
dbDisconnect(conn)
displayError("Error in Data Fetch from Database. ZERO ROWS
Returned. Quitting")
stop(safeError(NULL))

}

dbDisconnect(conn)

v_temp <- lat_lng(v_temp) %>%
mutate(tweet_text = TEXT) %>%
mutate(tweet_language = LANG) %>%
mutate(tweet_source = SOURCE) %>%
mutate(tweet_user = NAME) %>%
mutate(tweet_country = COUNTRY_CODE) %>%
mutate(tweet_place = PLACE_FULL_NAME) %>%
mutate(tweet_hashtag = HASHTAGS) %>%
mutate(tweet_followers_count = FOLLOWERS_COUNT) %>%
mutate(tweet_friends_count = FRIENDS_COUNT) %>%
mutate(tweet_urls_expanded_url = STATUS_URL) %>%
mutate(tweet_place_url = PLACE_URL) %>%
mutate(tweet_created_at = CREATED_AT) %>%
mutate(latitude = LAT) %>%
mutate(longitude = LNG)
},
```

```

error = function(e) {
# return a safeError
dbDisconnect(conn)
displayError(paste("Error in Data Fetch from Database. Quitting",
e))
stop(safeError(e))
}
)

return(v_temp)}
```

## EXTRACT

```
}
```

## EXTRAXT WORDS FROM THE READ TWEETS

```
extractWordsFromTweets <- function(p_tweet_data_frame) {
tryCatch(
{
# Remove the Emoticons
p_tweet_data_frame <- gsub("[^\x01-\x7F]", "", p_tweet_data_frame)
```

# Form the Corpus

```
v_corpus <- VCorpus(VectorSource(p_tweet_data_frame))
```

# Remove URLs

```
removeURL <- content_transformer(function(x) gsub(
"(f|ht)tp(s?)://\\S+", "", x, perl=T))
toSpace <- content_transformer(function (x, pattern) gsub(pattern, ",
x))
```

```

v_corpus <- tm_map(v_corpus, removeURL)
v_corpus <- tm_map(v_corpus, toSpace, "/")
v_corpus <- tm_map(v_corpus, toSpace, "@")
v_corpus <- tm_map(v_corpus, toSpace, "\\|")
v_corpus <- tm_map(v_corpus, toSpace, "'")
v_corpus <- tm_map(v_corpus, toSpace, ""))

# Create the Term Document Matrix after cleaning the Corpus
v_tdm <- TermDocumentMatrix(v_corpus,
control =


list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf))
)
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])

# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))
v_data <- subset(v_data, !(ind == "" || is.null(ind) || is.na(ind)))
},
error = function(e) {

```

```
displayError(paste("Error in Extracting Words from Tweets. This  
could be because the search does not produce any words in  
English Language. Please Try Again.", e, sep = " "))  
}  
)  
  
return(v_data)  
}
```

```
## EVALUATE EMOTIONS  
determineEmotionCount <- function(p_tidy_data) {  
v_combined <- sort(union(levels(p_tidy_data$word),  
levels(v_nrc$word)))  
v_text_sentiment2 <- inner_join(mutate(p_tidy_data,  
word=factor(word, levels=v_combined)),  
mutate(v_nrc, word=factor(word, levels=v_combined))  
)  
v_text_sentiment1 <- v_text_sentiment2 %>%  
count(word, index = row_number() %/% 80, sentiment) %>%  
mutate(v_original_n = n) %>%  
mutate(v_positive_negative = ifelse(sentiment %in%  
v_positive_emotions,  
'positive', 'negative')) %>%  
mutate(v_original_sentiment = sentiment) %>%  
spread(sentiment, n, fill = 0)  
v_text_sentiment <- v_text_sentiment1 %>%  
mutate(positive = ifelse("positive" %in%  
colnames(v_text_sentiment1), positive, 0)) %>%  
mutate(joy = ifelse("joy" %in% colnames(v_text_sentiment1), joy,  
0)) %>%
```

```

mutate(surprise = ifelse("surprise" %in%
colnames(v_text_sentiment1), surprise, 0)) %>%
mutate(trust = ifelse("trust" %in% colnames(v_text_sentiment1),
trust, 0)) %>%
mutate(anticipation = ifelse("anticipation" %in%
colnames(v_text_sentiment1), anticipation, 0)) %>%

mutate(negative = ifelse("negative" %in%
colnames(v_text_sentiment1), negative, 0)) %>%
mutate(anger = ifelse("anger" %in% colnames(v_text_sentiment1),
anger, 0)) %>%
mutate(disgust = ifelse("disgust" %in%
colnames(v_text_sentiment1), disgust, 0)) %>%
mutate(sadness = ifelse("sadness" %in%
colnames(v_text_sentiment1), sadness, 0)) %>%
mutate(fear = ifelse("fear" %in% colnames(v_text_sentiment1), fear,
0)) %>%
mutate(sentiment = (positive+joy+surprise+trust+anticipation) -
(negative+anger+disgust+sadness+fear)) %>%
ungroup()
return(v_text_sentiment)
}

```

```

## SAVE STATISTICS TO DATABASE
saveStatistics <- function() {
tryCatch(
{
v_statement <- paste("INSERT INTO TB_STATISTICS (",
"Date_of_Search,",

"Search_String,",

"No_requested,",

"No_of_tweets_fetched,",
```

```

"No_of_tweets_analysed,",
"No_of_hashtag_found,",
"No_of_countries,",
"No_of_places,",

"No_of_languages,",
"No_of_devices,",
"No_of_users,",
"Total_words,",
"Unique_words,",
"Prevalent_emotion)",

"VALUES (",
paste("", v_selected_date, "", sep = ""),
paste("", v_selected_subject, "", sep = ""),
paste(v_number_requested, "", sep = ""),
paste(v_number_fetched, "", sep = ""),
paste(v_number_analysed, "", sep = ""),
paste(v_number_hastag_found, "", sep = ""),
paste(v_number_countries, "", sep = ""),
paste(v_number_places, "", sep = ""),
paste(v_number_languages, "", sep = ""),
paste(v_number_devices, "", sep = ""),
paste(v_number_users, "", sep = ""),
paste(v_number_words, "", sep = ""),
paste(v_number_unique_words, "", sep = ""),
paste("", v_prevalent_emotion, "", sep = ""),
")", sep = " "
)

conn = dbConnect(jcc, jdbc_path, user=dsn_uid,
password=dsn_pwd)
if(!dbSendUpdate(conn, v_statement)) {
displayError("Error Writing to Database.")

```

```

} else {
dbCommit(conn)

}

dbDisconnect(conn)
},
error = function(e) {
dbDisconnect(conn)
}
)
}

#####
# READ THE SYSTEM PARAMETERS
#####

v_system_parameters <- read.csv(file = "SystemParameters.csv",
header = TRUE)
attach(v_system_parameters)

#####
# SET UP THE TWITTER ACCOUNT
#####

api_key <- as.character(v_system_parameters[(which(Parameter ==
"api_key")),]$Value)
api_secret_key <-
as.character(v_system_parameters[(which(Parameter ==
"api_secret_key")),]$Value)
access_token <- as.character(v_system_parameters[(which(Parameter ==
"access_token")),]$Value)
access_token_secret <-
as.character(v_system_parameters[(which(Parameter ==
"access_token_secret")),]$Value)

```

```

## Authenticate via Web Browser
token <- create_token(
  app = "ParthaEmotions",
  consumer_key = api_key,
  consumer_secret = api_secret_key,
  access_token = access_token,
  access_secret = access_token_secret)

#####
##### SET-UP THE DATABASE #####
#####

## Values for database connection
dsn_driver = as.character(v_system_parameters[(which(Parameter ==
"dsn_driver")),]$Value)
dsn_database = as.character(v_system_parameters[(which(Parameter ==
"dsn_database")),]$Value)
dsn_hostname = as.character(v_system_parameters[(which(Parameter ==
"dsn_hostname")),]$Value)
dsn_port = as.character(v_system_parameters[(which(Parameter ==
"dsn_port")),]$Value)
dsn_protocol = as.character(v_system_parameters[(which(Parameter ==
"dsn_protocol")),]$Value)
dsn_uid = as.character(v_system_parameters[(which(Parameter ==
"DBUserID")),]$Value)
dsn_pwd = as.character(v_system_parameters[(which(Parameter ==
"DBPassword")),]$Value)

## Connect to the Database
jcc = JDBC(dsn_driver, "Drivers/db2jcc4.jar", identifier.quote="");
jdbc_path = paste("jdbc:db2://", dsn_hostname, ":", dsn_port, "/",
  dsn_database, sep="");

```

```
##### SET UP THE GOOGLE MAPS
#####

google_maps_api_key =
as.character(v_system_parameters[(which(Parameter ==
"google_maps_api_key")),]$Value)

##### READ THE LEXICON
#####

v_nrc <- get_sentiments("nrc")
v_positive_emotions <- c("positive","joy","anticipation","surprise",
"trust")
v_negative_emotions <- c("negative","anger","disgust","fear",
"sadness")
v_positive_sentiment <- v_nrc %>%
filter(sentiment %in% v_positive_emotions)

##### SET UP INITIAL DATA TO DISPLAY
#####

# Fetch existing searches
v_existing_searches <- fetchExistingSearches()

# Fetch the Initial Set of Tweets from Database to Display
v_existing_subjects <- fetchExistingSubjects()

v_selected_subject <- v_existing_subjects[1,]$SEARCHSTRING
v_selected_date <- v_existing_subjects[1,]$DATEOFENQUIRY
```

```
v_tweets_to_display <- readTweets(v_selected_subject,
v_selected_date)

v_number_requested <- as.integer(o)
v_number_fetched <- as.integer(o)
v_number_analysed <- as.integer(o)
v_number_hashtag_found <- as.integer(o)
v_number_countries <- as.integer(o)
v_number_places <- as.integer(o)

v_number_languages <- as.integer(o)
v_number_devices <- as.integer(o)
v_number_users <- as.integer(o)
v_number_words <- as.integer(o)
v_number_unique_words <- as.integer(o)
v_prevalent_emotion <- ""

# Set up initial Emotion Analysis
tryCatch(
{
  v_words_in_tweets <- extractWordsFromTweets(
    (v_tweets_to_display %>%
      filter(tweet_language == 'en'))$tweet_text
  ) %>%
    rename(word = ind) %>%
    rename(n = values)
},
error = function(e) {
  # return a safeError
  displayError(paste("Error Setting up Chidiya. Quitting", e, sep =
  ""))
})
```

```
stop(safeError(e))
}

)

v_combined <- sort(union(levels(v_words_in_tweets$word),
levels(v_positive_sentiment$word)))
v_positive_sentiment_words <- semi_join(mutate(v_words_in_tweets,
word=factor(word, levels=v_combined)),
mutate(v_positive_sentiment, word=factor(word,
levels=v_combined))) %>%
count(word, sort = TRUE)

#####
# SET UP UI FOR THE APPLICATION
#####

ui <- fluidPage(
shinyjs::useShinyjs(),
theme = shinytheme("united"),
tags$head(
tags$style(
HTML(".shiny-notification {
position:fixed;
top: calc(50%);
left: calc(50%);
font-family:Verdana;
fontSize:xx-large;
}
"
)
)
),

```

```

# Application title
fluidRow(
  column(width = 3,
  img(src="ParthalnPetra.jpg", width=200, height=112),
  ),
  column(width = 6,
  h1("Chidiya", style="color:blue; text-align: center;"),

  h3("Tweet Analyser", style="color:darkred; text-align: center;"),
  ),
  column(width = 3,
  h4("...", style="color:blue; text-align: right;"),
  h4("Developed by: Partha Majumdar", style="color:blue; text-align: right;"),
  h5("Riyadh (Saudi Arabia), 29-February-2020", style="color:black;
  text-align: right;")
  )
),
# Sidebar with a slider input for number of bins
sidebarLayout(
  sidebarPanel(
    tabsetPanel(id = "search_option_panel", type = "tabs",
    tabPanel("Search String",
    h4("Enter Search String"),
    h5("Enter several subjects by separating with a comma"),
    "),
    h5("To search for hashtag, enter Search String starting with #. For
example: #Climate"),
   textInput(inputId = "v_search_string",
    label = "",
    value = ""
  )
)
)

```

```
placeholder = "Enter Subject(s) to Search"
),
sliderInput(inputId = "v_number_of_tweets",
label = h4("Number of Tweets to Fetch:"),
value = 750,
min = 500,
max = 5000
),
actionButton(inputId = "v_fetch", label = "Fetch")
),
tabPanel("Location",
h4("Enter Location"),
h5("Enter only ONE Location name"),
textInput(inputId = "v_search_location",
label = "",
value = "",
placeholder = "Enter Location to Search"
),
sliderInput(inputId = "v_number_of_tweets_location",
label = h4("Number of Tweets to Fetch:"),
value = 750,
min = 500,
max = 5000
),
actionButton(inputId = "v_fetch_location", label = "Fetch")
)
),
br(),
br(),
wellPanel(
h4("Past Searches", style="color:black; text-align: center;"),
```

```
plotOutput(outputId = "existingSearchesWordCloud")  
  
)  
,  
# Show a plot of the generated distribution  
mainPanel(  
tabsetPanel(id = "main_panel", type = "tabs",  
tabPanel("Raw Data",  
dataTableOutput(outputId = "tweets"),  
wellPanel(  
plotOutput(outputId = "tweetTimeSeries")  
)  
,  
tabPanel("Analysis",  
tabsetPanel(id = "analysis_panel", type = "tabs",  
tabPanel("World Map",  
fluidRow(  
div(id = "worldMapDisplay",  
column(width = 12,  
h3("Where are the people tweeting from",  
style= "color: blue; text-align: center;"),  
plotOutput (outputId = "WorldMap")  
)  
,  
div(id = "NoWorld Map", style="color: red; text-align: center;",  
img(src=" NoWorldMap.jpg", width=720, height=405)  
)  
,  
),  
tabPanel("Statistics",  
fluidRow(
```

```
div(id = "statisticsTab",
column(width = 3,
fluidRow(
  column(width = 12,
wellPanel(
  h5("Number of Tweets Analysed", style="color:black; text-align: center;"),
  h3(textOutput(outputId = "numberOfTweets"), style="color:blue; text-align: center;"),
  h6("Only Tweets with complete information are considered for analysis.", style="color:red; text-align: center;"),
)
),
  column(width = 12,
wellPanel(
  h5("Tweets from", style="color:black; text-align: center;"),
  h3(textOutput(outputId = "countryCount"), style="color:blue; text-align: center;"),
  h4("Country(ies)", style="color:blue; text-align: center;"),
  h6("See Map for details", style="color:red; text-align: center;"),
  dataTableOutput(outputId = "placeCount")
)
)
)

),
column(width = 5,
fluidRow(
column(width = 12,
wellPanel(
plotOutput(outputId = "languageCount")
)
)
```

```
),
  column(width = 12,
wellPanel(
  plotOutput (outputId = "sourceCount")
)
) #,
  )
),
column(width = 12,
fluidRow(
  column(width = 12,
wellPanel(
  h5("Tweets from", style="color:black; text-align: center;"),
  h3(textOutput(outputId = "userCount"), style="color:blue; text-align: center;"),
  h4("User(s)", style="color:blue; text-align: center;"),
  dataTableOutput(outputId = "mostTweeter")
)
),
column(width = 12,
h5("Hashtags Used", style="color:black; text-align: center;"),
plotOutput(outputId = "hashTagWordCloud")
)
)
),
div(id = "NoStatistics", style="color:red; text-align: center;",
img(src="NoStatistics.jpg", width=720,
  height=405)
)
)
```

```
),
tabPanel("Emotion Analysis",
  h3("Only Tweets in English Language considered for analysis",
    style="color:blue; text-align: center;"),
  div(id = "OutputTabsEmotionAnalysis",
    tabsetPanel(id = "emotion_analysis_panel", type = "tabs",
      tabPanel("Summary",
        fluidRow(
          column(width = 2,
            fluidRow(
              column(width = 12,
                wellPanel(
                  h5("Total Word Analysed", style="color:black; text-align: center;"),
                  h3(textOutput(outputId = "totalWordCount"), style="color:blue; text-align: center;"))
                )
              ),
            column(width = 12,
              wellPanel(
                h5("Unique Words Found", style="color:black; text-align: center;"),
                h3(textOutput(outputId =
"uniqueWordCount"), style="color:blue; text-align: center;"))
              )
            ),
          column(width = 12,
            wellPanel(
              h5("Words with Positive Emotions Found", style="color:black; text-align: center;"),
              h3(textOutput(outputId="positive
Sentiment WordCount"), style="color:blue;
text-align: center;"))
            )
          )
        )
      )
    )
  )
) ,
```

```
)  
)  
,  
column(width = 3,  
       wellPanel(  
         h5("Unique Words in Tweets", style="color:black; text-align:  
center;"),  
         dataTableOutput(outputId = "uniqueWords")  
       )  
,  
  
column(width 7,  
       fluidRow(  
         column(width = 12,  
                wellPanel(  
                  h5("Most Prevalent Emotion in Tweets", style="color:black; text-  
align: center;"),  
                  h2(textOutput(outputId = "mostPrevalent Emotion"),  
style="color:red; text-align: center;")  
                )  
              )  
,  
          fluidRow(  
            column(width = 4,  
                   plotOutput(outputId = "boxPlot")  
                 ),  
            column(width = 8,  
                   h5("Words Used", style="color:black; text-align: center;"),  
                   plotOutput(outputId = "wordWordCloud")  
                 )  
           )
```

```

)
)
),
tabPanel(
title = "Sentiment Analysis",
fluidRow(
column(width = 6,
h3("This graph indicates the prevalent emotions in the text."),

plotOutput(outputId = "sentiments")
),
column(width = 6,
h3("This graph indicates the volume of positive and negative
sentiments used through different words used in the Tweets. The
sentiments are displayed from the First to the Last Tweet
analysed."), analyzed."),

h4("Higher volume of GREEN color in the graph indicates
positivity in the text.", style="color:darkgreen; text-align: left;"),
h4("Higher volume of RED indicates negativity in the text.",
style="color:red; text-align: left;"),
plotOutput(outputId = "positiveNegative
Sentiments")
)
),
tabPanel("Word Analysis",
fluidRow(
column(width = 6,
sliderInput(inputId = "v_top_n_word_contribution",
h4("Set Top N Words to Include"),
min = 4, max = 200,

```

```
value = 10, step = 2, ticks = TRUE),
h4("This section highlights the words which have contributed to
the various emotions extracted in the text."),
h4("The emotions extracted are as per the lexicon used. The
lexicon assigns an emotion to the words as per generic
computation preset in the knowledge base."),

h4("While evaluating this graph, specific context needs to be kept
in mind before drawing conclusions."),
plotOutput(outputId = "wordContributionTo Sentiments")
),
column(width = 6,
sliderInput(inputId = "v_top_n_sentiment_score",
h4("Set Top N Words to Include"),
min = 4, max = 200,
value = 10, step = 2, ticks = TRUE),
h4("This graph displays the different words that have contributed
to the different emotions evaluated in the text."),
h3("The same word can contribute to different emotions based on
the context it is used in."),
plotOutput(outputId = "sentimentScore")
)
)
)
)
),
div(id = "OutputTabsNoEmotionAnalysis", style="color:red; text-align:
center;",
img(src="NoEmotionAnalysis.jpg", width=720, height=405)
)
)
)
```

```
),
tabPanel("Known Errors",

h3("1. When the 'Fetch Tweets' Operations fetched no Tweets, then  
the Display in the Analysis is all incorrect. 'Chidiya' does not  
become operational. An Error Message is displayed. And  
'Chidiya' allows for the next fetch operation to be conducted.",  
style="color:red; text-align: left;"),

h3("2. If the 'Fetch Tweets' Operations are used too frequently  
for large number of Tweets, 'Chidiya' can misbehave. This is  
because of limitation of number of Tweets that can be pulled  
from Twitter. The solution is to invoke 'Chidiya' after a gap of 15-  
30 minutes and normal service is restored.", style="color:blue; text-  
align: left;"),

h3("3. Progress Bar not synchronized with actual progress of  
task.", style="color:red; text-align: left;"),

h3("4. If there is a problem with the Internet connection, 'Chidiya'  
cannot connect to the Twitter API. 'Chidiya' detects this. However,  
'Chidiya' cannot presently recover from this error. To recover from  
this error, Internet connection has to be restored and 'Chidiya' has  
to be revoked once again.", style="color:blue; text-align: left;")

)
)
)
)
)
```

```
#####
# SET UP SERVER FOR THE
# APPLICATION
#
server <- function(input, output, session) {
  observeEvent(input$v_fetch_location,
  {
```

```
if(!(input$v_search_location == "" ||
is.null(input$v_search_location))) {
tryCatch(
{
withProgress(message = 'Fetching Tweets for Location',
  detail = 'This may take a while...', value = 100, {
v_tweets_to_display <-
v_tweets_for_location(input$v_search_location,
input$v_number_of_tweets_location)
extractWords()
})
refreshDisplay()
# Save Statistics
saveStatistics()
},
error = function(e) {
displayError(paste("Error Fetching Tweets. This could be because
the search does not fetch any Tweet OR the fetched Tweets do
not contain any Words in English Language OR the search was
for too few numbers of Tweets. The Graphs and other Outputs
will not be updated due to this. Try fetching a larger number of
Tweets OR using Search Terms in English Language.", e, sep =
"))
}
)
# Reset Display to default TAB
updateTabsetPanel(session, "main_panel",
selected = "Raw Data"
)
}
```

```

}

)

observeEvent(input$v_fetch,
{
if(!(input$v_search_string == "" || is.null(input$v_search_string))) {
tryCatch(
{
withProgress(message = 'Fetching Tweets for Search String',
  detail = 'This may take a while...', value = 100, {
v_tweets_to_display <- v_tweets(input$v_search_string,
input$v_number_of_tweets)
extractWords()
})
refreshDisplay()
# Save Statistics
saveStatistics()
},
error = function(e) {
displayError(paste("Error Fetching Tweets. This could be because
the search does not fetch any Tweet OR the fetched Tweets do
not contain any Words in English Language OR the search was
for too few numbers of Tweets. The Graphs and other Outputs
will not be updated due to this. Try fetching a larger number of
Tweets OR using Search Terms in English Language.", e, sep =
"))
}
}

# Reset Display to default TAB
updateTabsetPanel(session, "main_panel",
selected = "Raw Data"
)

```

```

}
}
)
)
extractWords <- function() {
tryCatch(
{
shinyjs::hide("OutputTabsEmotionAnalysis")
shinyjs::show("OutputTabsNoEmotionAnalysis")
v_words_in_tweets <- extractWordsFromTweets(
(v_tweets_to_display %>%
filter(tweet_language == 'en'))$tweet_text
) %>%
rename(word = ind) %>%
rename(n = values)
shinyjs::show("OutputTabsEmotionAnalysis")
shinyjs::hide("OutputTabsNoEmotionAnalysis")
},
error = function(e) {

```

displayError(paste("Error Extracting Words from Tweets. This could  
be because the search does not fetch any Words in English  
Language OR the search for too few number of Tweets. The  
Graphs and other Outputs will not be updated due to this. Try  
fetching a larger number of Tweets OR using Search Terms in  
English Language.", e, sep = " "))

```

}
)
```

```
v_combined <- sort(union(levels(v_words_in_tweets$word),
levels(v_positive_sentiment$word)))
v_positive_sentiment_words <-
semi_join(mutate(v_words_in_tweets, word=factor(word,
levels=v_combined)),
```

```

mutate(v_positive_sentiment, word=factor(word,
levels=v_combined))) %>%
count(word, sort = TRUE)
}
refreshDisplay <- function() {
# Dummy
determineLanguageCount(FALSE)
determineSourceCount(FALSE)
determineUserCount()
determineCountryCount()
determinePlaceCount(FALSE)
showUniqueWords(FALSE)
showMostPrevalentEmotion()
# Render on Fetch
output$tweets <- renderDataTable({showTweets()})
output$tweetTimeSeries <- renderPlot({showTweetTimeSeries()})
output$WorldMap <- renderPlot({showWorldMap()})
output$numberOfTweets <- renderText({nrow(v_tweets_to_display)})
```

```

output$languageCount <- renderPlot({determineLanguageCount()})
output$sourceCount <- renderPlot({determineSourceCount()})
output$userCount <- renderText({determineUserCount()})
output$countryCount <- renderText({determineCountryCount()})
output$placeCount <- renderDataTable({determinePlaceCount()})
output$mostTweeter <- renderDataTable({determineMostTweeter()})
output$hashTagWordCloud <-
renderPlot({showHashTagWordCloud()})
output$totalWordCount <-
renderText({return(sum(v_words_in_tweets$n))})
output$uniqueWordCount <-
renderText({return(nrow(v_words_in_tweets))})
```

```
output$positiveSentimentWordCount <-
renderText({return(nrow(v_positive_sentiment_words))})
output$uniqueWords <- renderDataTable({showUniqueWords()})
output$boxPlot <- renderPlot({showBoxPlot()})
output$wordWordCloud <- renderPlot({showWordWordCloud()})
output$wordContributionToSentiments <-
renderPlot({showWordContributionToSentiments()})
output$sentiments <- renderPlot({showSentiments()})
output$positiveNegativeSentiments <-
renderPlot({showPositiveNegativeSentiments()})

output$mostPrevalentEmotion <-
renderText({showMostPrevalentEmotion()})
output$sentimentScore <- renderPlot({determineSentimentScore()})
}

v_tweets <- function(p_search_string, p_number_of_tweets) {
tryCatch(
{
shinyjs::hide("OutputTabsEmotionAnalysis")
shinyjs::show("OutputTabsNoEmotionAnalysis")
shinyjs::hide("statisticsTab")
shinyjs::show("NoStatistics")
shinyjs::hide("worldMapDisplay")
shinyjs::show("NoWorldMap")
v_temp <- search_tweets(
p_search_string,
'-filter' = "replies",
n = p_number_of_tweets,
include_rts = FALSE,
retryonratelimit = FALSE #,
)
}
```

```
v_temp <- gatherTweetData(v_temp, p_search_string,
p_number_of_tweets)
shinyjs::show("OutputTabsEmotionAnalysis")
shinyjs::hide("OutputTabsNoEmotionAnalysis")
shinyjs::show("statisticsTab")
shinyjs::hide("NoStatistics")
shinyjs::show("worldMapDisplay")
shinyjs::hide("NoWorldMap")
},
error = function(e) {
displayError(paste("Error Fetching Tweets.", e))
}
)
return(v_temp)
}
v_tweets_for_location <- function(p_search_location,
p_number_of_tweets) {
tryCatch(
{
shinyjs::hide("OutputTabsEmotionAnalysis")
shinyjs::show("OutputTabsNoEmotionAnalysis")
shinyjs::hide("statisticsTab")
shinyjs::show("NoStatistics")
shinyjs::hide("worldMapDisplay")
shinyjs::show("NoWorldMap")
v_temp <- search_tweets(
geocode = lookup_coords(p_search_location, apikey =
google_maps_api_key),
'-filter' = "replies",
n = p_number_of_tweets,
include_rts = FALSE,
```

```
retryonratelimit = FALSE #,
)
v_temp <- gatherTweetData(v_temp, p_search_location,
p_number_of_tweets)
shinyjs::show("OutputTabsEmotionAnalysis")
shinyjs::hide("OutputTabsNoEmotionAnalysis")
shinyjs::show("statisticsTab")

shinyjs::hide("NoStatistics")
shinyjs::show("worldMapDisplay")
shinyjs::hide("NoWorldMap")
},
error = function(e) {
displayError(paste("Error Fetching Tweets.", e))
}
)
return(v_temp)
}

gatherTweetData <- function(p_tweets, p_search_string,
p_number_of_tweets) {
v_number_fetched <- nrow(p_tweets)
v_temp <- lat_lng(p_tweets) %>%
filter(lat != "NA") %>%
filter(lng != "NA") %>%
mutate(tweet_text = text) %>%
mutate(tweet_language = lang) %>%
mutate(tweet_source = source) %>%
mutate(tweet_user = name) %>%
mutate(tweet_country = country_code) %>%
mutate(tweet_place = place_full_name) %>%
mutate(tweet_hashtag = hashtags) %>%
mutate(tweet_followers_count = followers_count) %>%
```

```
mutate(tweet_friends_count = friends_count) %>%
mutate(tweet_urls_expanded_url = status_url) %>%
mutate(tweet_place_url = place_url) %>%
mutate(tweet_created_at = created_at) %>%
mutate(latitude = lat) %>%

mutate(longitude = lng)
v_number_analysed <- nrow(v_temp)
v_dataframe_to_write <- v_temp
v_dataframe_to_write$SEARCHSTRING <- p_search_string
v_dataframe_to_write$DATEOFENQUIRY <- format(Sys.Date(), "%Y-%m-%d")
v_selected_subject <- p_search_string
v_number_requested <- p_number_of_tweets
v_selected_date <- format(Sys.Date(), "%Y-%m-%d")
write_as_csv(
  v_dataframe_to_write,
  file_name = "last_fetched.csv",
  prepend_ids = TRUE,
  na = "",
  fileEncoding = "UTF-8"
)
return(v_temp)
}
output$totalWordCount <- renderText({
  shinyjs::hide("OutputTabsNoEmotionAnalysis")
  return(sum(v_words_in_tweets$n))
})
output$uniqueWordCount <- renderText({
  return(nrow(v_words_in_tweets))
})
output$positiveSentimentWordCount <- renderText({
```

```
return(nrow(v_positive_sentiment_words))
})
output$uniqueWords <- renderDataTable({
  showUniqueWords()
})
output$boxPlot <- renderPlot({
  showBoxPlot()
})
output$tweets <- renderDataTable({
  showTweets()
})
output$WorldMap <- renderPlot({
  showWorldMap()
})
output$numberOfTweets <- renderText({
  shinyjs::hide("NoStatistics")
  nrow(v_tweets_to_display)
})
output$languageCount <- renderPlot({
  determineLanguageCount()
})
output$sourceCount <- renderPlot({
  determineSourceCount()
})
output$userCount <- renderText({
  determineUserCount()
})
output$mostTweeter <- renderDataTable({
  determineMostTweeter()
})
output$countryCount <- renderText({
```

```
determineCountryCount()

})

output$placeCount <- renderDataTable({
determinePlaceCount()
})

output$hashTagWordCloud <- renderPlot({
showHashTagWordCloud()
})

output$wordWordCloud <- renderPlot({
showWordWordCloud()
})

showBoxPlot <- function() {
boxplot(v_words_in_tweets$n,
main = "Word Distribution",
xlab = NULL,
ylab = "Frequency",
col = "orange",
border = "brown",
horizontal = FALSE,
notch = FALSE
)
}

determineLanguageCount <- function(flag = TRUE) {
v_temp_df <- v_tweets_to_display %>%
ungroup() %>%
mutate(fac_language = as.factor(tweet_language))
v_temp <-
as.data.frame(table(fct_explicit_na(v_temp_df$fac_language)))
if(nrow(v_temp) > 0) {
v_temp <- v_temp %>%
```

```

mutate(Language = reorder(Var1, Freq))
} else {
v_temp <- data.frame(Language = character(),
Freq = integer(),
stringsAsFactors = FALSE)
}
v_data <- v_temp$Freq
v_labels <- paste(paste(v_temp$Language, "\n",
round(v_data/sum(v_data)*100, 2), sep=" "),
"%", sep = ""))
v_number_languages <- nrow(v_temp)
if(flag) {
pie(v_data, labels = v_labels, main = "Tweet Languages")
}
}
determineSourceCount <- function(flag = TRUE) {
v_temp_df <- v_tweets_to_display %>%
ungroup() %>%
mutate(fac_source = as.factor(tweet_source))
v_temp <-
as.data.frame(table(fct_explicit_na(v_temp_df$fac_source)))
if(nrow(v_temp) > 0) {
v_temp <- v_temp %>%
mutate(Source = reorder(Var1, Freq))
} else {
v_temp <- data.frame(Source = character(),
Freq = integer(),
stringsAsFactors = FALSE)
}

v_data <- v_temp$Freq
v_labels <- paste(paste(v_temp$Source, "\n",

```

```

round(v_data/sum(v_data)*100, 2), sep=" "),
"%", sep = "")
v_number_devices <- nrow(v_temp)
if(flag)  {
  pie(v_data, labels = v_labels, main = "Tweet Source")
}
}
determineUserCount <- function() {
  v_temp_df <- v_tweets_to_display %>%
    ungroup() %>%
    mutate(fac_user = as.factor(tweet_user))
  v_temp <- as.data.frame(table(fct_explicit_na(v_temp_df$fac_user)))
  if(nrow(v_temp) <= 0) {
    v_temp <- data.frame(Var1 = character(),
      Freq = integer(),
      stringsAsFactors = FALSE)
  }
  v_number_users <- nrow(v_temp)
  return(nrow(v_temp))
}
determineCountryCount <- function() {
  v_temp_df <- v_tweets_to_display %>%
    ungroup() %>%
    mutate(fac_country = as.factor(tweet_country))
  v_temp <-
  as.data.frame(table(fct_explicit_na(v_temp_df$fac_country)))

  if(nrow(v_temp) <= 0) {
    v_temp <- data.frame(Var1 = character(),
      Freq = integer(),
      stringsAsFactors = FALSE)
  }
}

```

```

v_number_countries <- nrow(v_temp)
return(nrow(v_temp))
}
determinePlaceCount <- function(flag = TRUE) {
v_temp_df <- subset(v_tweets_to_display, !(tweet_place == "" ||
is.null(tweet_place) || is.na(tweet_place)) ||
(tweet_place_url == "" || is.null(tweet_place_url) ||
is.na(tweet_place_url)))) %>%
ungroup() %>%
mutate(fac_place = as.factor(tweet_place))
v_temp <- as.data.frame(table(fct_explicit_na(v_temp_df$fac_place)))
if(nrow(v_temp) > 0) {
v_temp <- v_temp %>%
mutate(Place = reorder(Var1, Freq)) %>%
filter(Place != "(Missing)")
# Determine Place URL
v_temp$PlaceURL <- ""
for (i in 1:nrow(v_temp)) {
v_temp[i,]$PlaceURL <-
head(v_tweets_to_display[which(v_tweets_to_display$tweet_place ==
v_temp[i,]$Place),], 1)$tweet_place_url
}
v_temp <- v_temp %>%
mutate(place_with_link = paste("href='",
PlaceURL,
"",
target='_blank'>",
Place,
"",
sep = ""))
}
} else {

```

```

v_temp <- data.frame(place_with_link = character(),
                      Freq = integer(),
                      stringsAsFactors = FALSE)
}

v_number_places <- nrow(v_temp)
if(flag) {
  DT::datatable(
    {v_temp[,c("place_with_link", "Freq")]},
    colnames = c("Place", "#"),
    escape = FALSE,
    rownames = FALSE,
    options = list(
      fixedColumns = TRUE,
      autoWidth = FALSE,
      ordering = TRUE,
      pageLength = 6,
      dom = 'tip'
    ))
}
}

determineMostTweeter <- function() {

  v_temp_df <- v_tweets_to_display %>%
    ungroup() %>%
    mutate(fac_user = as.factor(tweet_user))
    <- as.data.frame(table(fct_explicit_na(v_temp_df$fac_user)))
  if(nrow(v_temp) > 0) {
    v_temp <- v_temp %>%
      mutate(Tweeter = reorder(Var1, Freq)) %>%
      filter(Tweeter != "(Missing)")
      # Determine number of Followers and Friends
      v_temp$Followers <- as.integer(0)
  }
}

```

```

v_temp$Friends <- as.integer(0)
for (i in 1:nrow(v_temp)) {
  v_temp[i,]$Followers <- as.integer(max(subset(v_tweets_to_display,
    tweet_user == v_temp[i,]$Tweeter)$tweet_followers_count))
  v_temp[i,]$Friends <- as.integer(max(subset(v_tweets_to_display,
    tweet_user == v_temp[i,]$Tweeter)$tweet_friends_count))
}
} else {
  v_temp <- data.frame(Tweeter = character(),
  Freq = integer(),
  Followers = integer(),
  Friends = integer(),
  stringsAsFactors = FALSE)
}
DT::datatable(
  {v_temp[,c("Tweeter", "Freq", "Followers", "Friends")]},
  colnames = c("Tweeter", "#", "Followers", "Friends"),
  escape = TRUE,
  rownames = FALSE,
  options = list(
    fixedColumns = TRUE,
    autoWidth = FALSE,
    ordering = TRUE,
    pageLength = 3,
    dom = 'tip'
  ))
}

showHashTagWordCloud <- function() {
  tryCatch(
  {

```

```

v_corpus <-
Corpus(VectorSource(v_tweets_to_display$tweet_hashtag))
# Create the Term Document Matrix after cleaning the Corpus
v_tdm <- TermDocumentMatrix(v_corpus,
control =
list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf)))
)
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)

# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])
# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))
v_number_hashtag_found <- nrow(v_data)
withProgress(message = 'Generating Word Cloud',
detail = 'This may take a while...', value = 0, {
suppressWarnings(
wordcloud(words = v_data$ind, freq = v_data$values,
min.freq = 1,
max.words = 100,
random.order=FALSE,
colors=brewer.pal(8, "Dark2")
)
)
})

```

```

},
error = function(e) {
displayError(paste("Error creating HashTag Word Cloud", e))
}
)
}

output$existingSearchesWordCloud <- renderPlot({
tryCatch(
{

v_corpus <-
Corpus(VectorSource(v_existing_searches$SEARCH_STRING))
# Create the Term Document Matrix after cleaning the Corpus
v_tdm <- TermDocumentMatrix(v_corpus,
control =
list(removePunctuation = TRUE,
stopwords = TRUE,
tolower = TRUE,
stemming = TRUE,
removeNumbers = TRUE,
bounds = list(global = c(1, Inf)))
)
)

# Find the Frequent Words
v_ft <- findFreqTerms(v_tdm, lowfreq = 1, highfreq = Inf)
# Convert to a Matrix
v_matrix <- as.matrix(v_tdm[v_ft,])
# Sort the Words in DESCENDING Order and create Data Frame
v_data <- stack(sort(apply(v_matrix, 1, sum), decreasing = TRUE))
v_number_hashtag_found <- nrow(v_data)
withProgress(message = 'Generating Word Cloud',
detail = 'This may take a while...', value = 0, {

```



```
withProgress(message = 'Tabulating Tweets',
detail = 'This may take a while...', value = 100, {

v_temp <- v_tweets_to_display %>%
mutate(tweet_with_link = paste("href=\"",
  tweet_urls_expanded_url,
\"", target='_blank'>",
  tweet_text,
"",
sep = ""))
)

DT::datatable(
{select(v_temp, tweet_with_link)},
colnames = c('Tweet Text (Click on the Tweet to see the Original
Tweet)'),
caption = htmltools::tags$caption(
style = 'caption-side: bottom; text-align: center;',
'Table 1: ',
htmltools::em('Fetched Tweets')
),
extensions = 'Buttons',
escape = FALSE,
options = list(
fixedColumns = TRUE,
autoWidth = FALSE,
ordering = TRUE,
pageLength = 10,
dom = 'Bftsp',
buttons = c('copy', 'csv', 'excel')
))
})
```

```

output$tweetTimeSeries <- renderPlot({
  showTweetTimeSeries()
})

showTweetTimeSeries <- function() {
  # Save current locale
  loc <- Sys.getlocale("LC_TIME")
  # Set Locale
  Sys.setlocale("LC_TIME", "C")
  # Convert to POSIXct
  v_tweets_to_display$tweet_created_at <-
    as.POSIXct(v_tweets_to_display$tweet_created_at, '%Y-%m-%d
%H:%M:%S', tz = Sys.timezone())
  # Then set back to the old locale
  Sys.setlocale("LC_TIME", loc)
  ## Plot time series of tweets frequency
  ts_plot(v_tweets_to_display[, c("tweet_created_at")], "mins") +
    ggplot2::theme_minimal() +
    ggplot2::theme(plot.title = ggplot2::element_text(face = "bold")) +
    ggplot2::labs(
      x = NULL, y = NULL,
      title = paste("Frequency of", v_selected_subject, "Twitter statuses",
      sep = " "),
      subtitle = "Twitter status (tweet) counts aggregated using one-
      minute intervals",
      caption = "\nSource: Data collected from Twitter's REST API via
      rtweet"
    )
}

showUniqueWords <- function(flag = TRUE) {
  v_number_words <- sum(v_words_in_tweets$n)
}

```

```

v_number_unique_words <- nrow(v_words_in_tweets)
if(flag) {
withProgress(message = 'Tabulating Unique Words',
detail = 'This may take a while...', value = 100, {
DT::datatable(
{v_words_in_tweets[, c("word", "n")]},
colnames = c('Word', '#'),
caption = htmltools::tags$caption(
style = 'caption-side: bottom; text-align: center;',
'Table 4: ',
htmltools::em('Unique Words')
),
extensions = 'Buttons',
escape = TRUE,
rownames = FALSE,
options = list(
fixedColumns = TRUE,
autoWidth = FALSE,
ordering = TRUE,
pageLength = 5,
dom = 'Bftsp',
buttons = c('copy', 'csv', 'excel')
))
})
}
}
}

```

```
showWorldMap <- function() {  
  shinyjs::hide("NoWorldMap")  
  withProgress(message = 'Generating Map',  
               detail = 'This may take a while...', value = 100, {  
    par(mar = c(0, 0, 0, 0))
```

```

map('world', fill = FALSE, col = 1:10, wrap=c(-180,180))
## plot lat and lng points onto state map
with(v_tweets_to_display, points(lng, lat, pch = 21, cex = .75, col =
rgb(1, 0, 0, 1)))
})
}
output$wordContributionToSentiments <- renderPlot({
showWordContributionToSentiments()
})
showWordContributionToSentiments <- function() {
v_data <- determineEmotionCount(v_words_in_tweets) %>%
filter(row_number() < input$v_top_n_word_contribution) %>%
group_by(v_original_sentiment) %>%
ungroup() %>%
mutate(word = reorder(word, v_original_n))
v_data$word <- as.character(v_data$word)
v_data <- data.frame(apply(v_data, 2, unclass))
ggplot(aes(x = word, y = v_original_n, fill = v_original_sentiment),
data = v_data, scale_x_continuous(breaks = NULL)) +
geom_col(show.legend = FALSE) +
facet_wrap(~v_original_sentiment, scales = "free_y") +

```

labs(y = "Words Contribution to Individual Sentiments", x = NULL) +

```

coord_flip()
}
output$sentiments <- renderPlot({
showSentiments()
})
showSentiments <- function() {
v_emotions <- determineEmotionCount(v_words_in_tweets) %>%
select(index, anger, anticipation, disgust, fear, joy,

```

```

    sadness, surprise, trust) %>%
melt(id = "index") %>%
rename(linenumber = index, sentiment_name = variable, value =
value)
v_emotions_group <- group_by(v_emotions, sentiment_name)
v_by_emotions <- summarise(v_emotions_group,
values=sum(value))
ggplot(aes(reorder(x=sentiment_name, values), y=values,
fill=sentiment_name), data = v_by_emotions) +
geom_bar(stat = 'identity') +
ggtitle('Sentiment in Tweets') +
coord_flip() +
theme(legend.position="none")
}
output$positiveNegativeSentiments <- renderPlot({
showPositiveNegativeSentiments()
})
showPositiveNegativeSentiments <- function() {

v_data <- determineEmotionCount(v_words_in_tweets) %>%
mutate(v_original_n = ifelse(v_positive_negative == "negative",
-v_original_n, v_original_n))
ggplot(data = v_data, aes(x = index, y = v_original_n, fill =
v_positive_negative)) +
geom_bar(stat = 'identity', position = position_dodge()) +
theme_minimal() +
ylab("Sentiment") +
ggtitle("Positive and Negative Sentiment in Tweets") +
scale_color_manual(values = c("red", "dark green")) +
scale_fill_manual(values = c("red", "dark green"))
}
output$mostPrevalentEmotion <- renderText({

```

```

showMostPrevalentEmotion()
})
showMostPrevalentEmotion <- function() {
v_emotions <- determineEmotionCount(v_words_in_tweets) %>%
select(index, anger, anticipation, disgust, fear, joy,
sadness, surprise, trust) %>%
melt(id = "index") %>%
rename(linenumber = index, sentiment_name = variable, value =
value)
v_emotions_group <- group_by(v_emotions, sentiment_name)
v_by_emotions <- summarise(v_emotions_group, values =
sum(value))
v_temp <- v_by_emotions %>%
mutate_if(is.factor, as.character)
v_prevalent_emotion <- ifelse(nrow(subset(v_temp, values ==
max(values))) == 1,
unname(unlist(subset(v_temp, values == max(values)))),
"No Prevalent Emotion")
return(v_prevalent_emotion)
}
output$sentimentScore <- renderPlot({
determineSentimentScore()
})
determineSentimentScore <- function() {
v_combined <- sort(union(levels(v_words_in_tweets$word),
levels(v_nrc$word)))
v_word_count <- inner_join(mutate(v_words_in_tweets,
word=factor(word, levels=v_combined)),
mutate(v_nrc, word=factor(word, levels=v_combined)))
) %>%
count(word, sentiment, sort = TRUE)
}

```

```
v_data <- v_word_count %>%
  filter(row_number() < input$v_top_n_sentiment_score) %>%
  mutate(n = ifelse(sentiment %in% v_negative_emotions, -n, n))
%>%
  mutate(word = reorder(word, n))
ggplot(aes(word, n, fill = sentiment), data = v_data) +
  geom_col() +
  coord_flip() +
  labs(y = "Sentiment Score")
}
}

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

## Conclusion

In this chapter, we discussed the code for developing a data product for emotion analysis on Twitter data using R programming called Chidiya. We started by discussing the features of Chidiya that we would develop. Then, we went through piece by piece developing each component of Chidiya. In the end, we put all the components together to present the complete code of Chidiya.

We only discussed features for analyzing Twitter data fetched for a search string and Twitter data from a location. There are many more combinations of Twitter data that we can fetch and/or filter. For example, we may want to fetch Twitter data posted by a person, or we may want to fetch Twitter data for posts which have more than x number of likes/retweets/replies, etc.

As we have discussed all the essential concepts for developing data products using R programming and introduced many programming constructs and techniques, I would expect that you would develop a much more advanced version of Chidiya. Also, I would expect that you would delve deeper into data analytics and conduct much more advanced analysis of Twitter data. This will help you explore emotion analysis.

### Points to remember

Always find common pieces of code in an application and create functions. This makes the code of the application modular and reduces the amount of code. This also lends to the code to be more maintainable. When a piece of code can be used across multiple applications, create a library with such code snippets and include these libraries in the applications.

Appropriate database drivers should be installed on the machine for the Shiny application to be able to interact with the database.

### Multiple choice questions

The function to close the database connection in the RJDBC library is:

dbDisconnect

dbClose

Both of these

None of these

The set\_config() function, used to also set the HTTP version, is available in which library?

http

httr

configuration

None of these

The function to create a modal window in a Shiny application is:

Dialog

dialog

modalDialog

None of these

The function to abort a Shiny application is:

abort

stop

Both of these

None of these

The dbSendQuery() function returns the column names in:

Upper case

Lower case

Both of these

None of these

## Answers to MCQs

A

B

C

B

A

## Questions

Add features to Chidiya to conduct an emotion analysis of tweet for a user.

Add features to Chidiya to conduct an **emotion analysis** while streaming tweets.

## Key terms

**UI:** User Interface

**UX:** User experience.

## BONUS

### WhatsApp Chat Analysis

This is a bonus chapter to further enhance your understanding of data analysis and data visualization. In this chapter, I will discuss how to analyze chats exchanged on WhatsApp.

WhatsApp is a very popular means of communicating in the modern world using mobile phones and computers. WhatsApp has replaced the initial **short messaging service (SMS)** as available in the mobile phones. While SMS is still in use, its usage has vastly reduced as WhatsApp provides many more features and functionalities for exchanging messages. For example, it is not only possible to exchange text using WhatsApp but also to add graphics like emoticons in the messages and attachments.

WhatsApp is not only used for exchanging messages but also used as an online help desk by most service companies. Many companies and individuals use WhatsApp for purposes like project management, product advertisements, group communications, etc. Many companies have programmed robots which can interact with humans through WhatsApp for resolving queries.

WhatsApp can be used for voice and video communications. These services are restricted in some countries. However, these will not form a part of the analysis that we will discuss in this

chapter. We will restrict ourselves to the text messages exchanged using WhatsApp for our analysis.

We will begin the chapter by discussing how to download chats from WhatsApp. We will round up the chapter with a discussion on how to visualize the data of the chats. I leave it to you to conduct emotion analysis on the chat data downloaded from WhatsApp using the concepts discussed in the book.

During these discussions, I will also discuss some aspects of data visualization using the `ggplot2` library.

## Structure

In this chapter, we will discuss the following topics:

Introduction to WhatsApp

How to download data from WhatsApp?

- Downloading WhatsApp data from Apple iPhones
- Downloading WhatsApp data from Android phones

The structure of WhatsApp chat data

Reading WhatsApp chat data using R

Visualizing WhatsApp chat data

- Messages per day

- Messages per weekday

- Radar charts

- Messages per hour

- Heat map
- Messages per author
- Emoji analysis
- Word analysis

## Objectives

After studying this unit, you should be able to:

Understand how to download data from WhatsApp.

Visualize the data downloaded from WhatsApp.

## Introduction to WhatsApp

WhatsApp needs no introduction as it is so popular and widely used in the modern world. WhatsApp was created by *Jan Koum and Brian Acton on 24th February, 2009. Both Jan and Brian were former Yahoo! employees.* The first stable version of WhatsApp was released on Apple App Store in August 2009 and named WhatsApp 2.0. WhatsApp was launched for Android phones in August 2010. On 19th February, 2014, WhatsApp was acquired by Facebook for USD 19 billion.

*WhatsApp uses a customized version of Extensible Messaging and Presence Protocol* WhatsApp has more than 2 billion active users in more than 180 countries as of 2020.

## How to download data from WhatsApp?

WhatsApp stores all the data in servers. It is possible to access this data using APIs exposed by WhatsApp. However, we will not discuss this as it requires a higher level of programming knowledge. I will leave this aspect to the interested readers to explore on their own.

What is possible for everyone is to download data from WhatsApp using a feature provided in WhatsApp to export data regarding any chat. As individuals can only export their own chat data, there is no infringement of privacy.

We will discuss how to download WhatsApp chat data from Apple iPhones and Android phones. The feature to export WhatsApp chat data is not available in the WhatsApp desktop version.

## [\*\*Downloading WhatsApp data from Apple iPhones\*\*](#)

We will first take a step-by-step tour of how to download WhatsApp chat data from Apple iPhones:

### **Open WhatsApp on iPhone**

Locate the WhatsApp application on iPhone and invoke it. The following screenshots illustrate what the WhatsApp icon looks like:

4:49



Authenticator



HDFC Bank



Coursera



FindMy...



Gmail



Flipkart...



Amazon



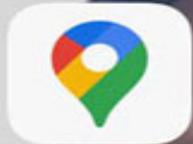
Gaana



Authenticator



ESPNcricinfo



Google Maps



Messenger



FreeCell



KTrack



LinkedIn Jobs



Ola Cabs



R Language



SMUDE



Solitaire Classic



SonyLIV



TOI



Domino's



HUAWEI HiLink



WhatsApp





**Figure 1:** What it looks on the iPhone.

## Locating the chat to analyze

When you open the WhatsApp application, you can locate all the chats in the Chats tab. The Chats tab can be invoked by clicking the Chat icon as shown in [figure 2](#) with the red color marker. In the Chats tab, click on any chat to see its contents. In [figure one](#) one chat is highlighted in green color:

4:49



Edit



# Chats

Search



Archived Chats

0

Broadcast Lists

New Group

**SS Apartment Residents**

4:15 AM

Prabhakar Anvekar:

Shubhodaya Suprabhat. Om Namah S...

**Hamoud Tamkeen**

Yesterday

You are welcome uncle

**Abdullah Alshammari**

Yesterday

**PGAILD- SEP INTAKE 2020**

Yesterday

+971 50 662 7344: Fine for me

**BPB 381 Partha Majumdar**

Yesterday

✓ You: Thanks

**T&S Marketing**

Saturday

✓✓ You: T&amp;S - Cloud Solutions - VIDA+ Migration - 20201205.pptx

**Amro**

Saturday

✓✓ You: Dear Abdullah, I am having



Status



Calls



Camera



Chats



Settings



***Figure 2: What it looks on the iPhone***

### **Invoking the actions available for a Chat Group**

Once you are in a chat, click on the group name as shown in [figure 3](#) in red color. This will lead to the options available from WhatsApp for different actions possible in the group. If the chat is with an individual, the individual's name would appear in the box marked with red color:

4:49



**SS Apartment Residents**

Akshar Kishore, Bidesh, Bineeta, Deepshre...



use this as an excuse for themselves.

All of you know best. It was just an opinion only and nothing more.

11:57 AM

Dear All,

Please note that Mr. Chatterji (TF01) has been communicating with Ranjeev (TF09) regarding all the stated issues for very long. He has not bothered to even respond till today morning.

12:03 PM ✓✓

I would like to inform all of you as I am helping Deepshree keep accounts, the amount in the Bank Account is very minimal. There is hardly any cash in hand. If you do not take initiatives to recover the dues, we will soon not be able to carry out essential activities for the apartment.

12:11 PM ✓✓

Today

**Prabhakar Anvekar SF01 Sneha Sindhu**



Shubhodaya Suprabhat.

Om Namah Shivaya.

Lord Shiva bless you and your family with good health peace and prosperity.

Wish you a nice peaceful and fruitful day and week ahead.

Go out only when required for essential work.

Prabhakar Anvekar.

4:15 AM





***Figure 3:*** What it looks on the iPhone

## Finding

Once we are in the menu containing different actions possible in a chat group, we need to locate the option for Exporting Data from WhatsApp. For this, scroll down through the options. You will find the Export Chat option as shown in [figure 4](#) in the red color box:

4:49

[Back](#)**Group Info**[Share](#)**Dipankar Chatterjee**

Admin

Amazing Me!

**Nadeem Ahmed FF01 Sneha Sindhu**

Admin

Urgent calls only

**+91 99454 76790**

Admin

~Laxmi Shetty M,TO,GITI,

**Akshar Kishore GF09 Sneha Sindhu**

You can have it all. You just can't have it all at once.

**Bidesh Tenant SF08 Sneha Sindhu**

We miss you...

**Bineeta TF08**

Available

**Dinesh Urs - TF07 Sneha Sindhu****Manohar FF10 Sneha Sindhu**

All is well



25 more

[Export Chat](#)[Clear Chat](#)[Exit Group](#)[Report Group](#)

Created by Nadeem Ahmed FF01 Sneha Sindhu.

Created 28 May 2019.



**Figure 4:** What it looks on the iPhone

### Exporting the chat data

WhatsApp provides an option to export the chat data with or without the attachments. For the analysis that we will discuss in this book, we do not need the attachments. So, we will choose the option to export the data without the attachments shown as follows:

4:49

[Back](#)**Group Info**[Share](#)**Dipankar Chatterjee**

Amazing Me!

Admin

**Nadeem Ahmed FF01 Sneha Sindhu**

Urgent calls only

Admin

**+91 99454 76790**

Admin

~Laxmi Shetty M,TO,GITI,

**Akshar Kishore GF09 Sneha Sindhu**

You can have it all. You just can't have it all at once.

**Bidesh Tenant SF08 Sneha Sindhu**

We miss you...

**Bineeta TF08**

Available

**Dinesh Urs - TF07 Sneha Sindhu****Manohar FF10 Sneha Sindhu**

All is well



25 more

**Export Chat**

Attaching media will generate a larger chat archive.

[Attach Media](#)[Without Media](#)[Cancel](#)



***Figure 5: What it looks on iPhone***

On clicking this option, WhatsApp exports the data and stores it in the iPhone.

### **Transferring the chat data to a computer**

Once WhatsApp completes exporting the data, it can be transferred to any computer. There are many options for this including sending by email, WhatsApp, etc. I will choose the option to transfer to my MacBook from my iPhone. In my case, both the devices are interconnected and thus, it is simpler.

4:50

[Back](#)**Group Info**[Share](#)**Dipankar Chatterjee**

Admin

Amazing Me!

**Nadeem Ahmed FF01 Sneha Sindhu**

Admin

Urgent calls only

**+91 99454 76790**

Admin

~Laxmi Shetty M,TO,GITI,

**Akshar Kishore GF09 Sneha Sindhu**

You can have it all. You just can't have it all at once.

**Bidesh Tenant SF08 Sneha Sindhu**

We miss you...

**Bineeta TF08**

Available

**WhatsApp Chat - SS Apartment Reside...**

ZIP Archive · 84 KB

**PARTHAs**  
MacBook Pro**Hamoud**  
Tamkeen**Abdullah**  
Alshammari**SS Apartment**  
Residents**PGA**  
INTA

AirDrop



Webex Meet



Messages



Mail

[Copy](#)[Save to Files](#)

Save as Draft

W

*Figure 6: What it looks on iPhone*

On exporting the data file, it creates a ZIP file on the target. When the ZIP file is unzipped, it provides a TXT file containing all the chat data. This TXT file can be used for analysis.

## [\*Downloading WhatsApp data from Android phones\*](#)

The interface for downloading WhatsApp chat data on Android phones is similar to that of the interface on Apple iPhones. However, there are some subtle visual differences. So, I will just provide the images of the different steps:

15:18



Weather



File Manager



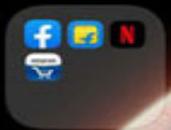
ShareMe\*



Music



Settings



Misc



Mi Remote



WhatsApp



Uber



Prime Video



LinkedIn



Udemy



Zoom



Lite



Mi Pay



MakeMyTrip



Hangouts



Webex Meet



Meet



JioMeet



*Figure 7: Step 1*

15:18



## WhatsApp

CHATS 12

STATUS

CALLS

**Gettu 4th March**

You left

16/07/2020

**Secure Lock Poc**

You left

16/07/2020

**DT-CS-Work**

Soumitra Banerjee added you

06/02/2020

**Pipeline Enhancements**

Soumitra Banerjee added you

06/02/2020

**HPCL Ops Team**

Abdul Raheem added you

06/02/2020

**Golden Circle Support**

Sweta Annectos left

06/06/2020

**+1 (773) 512-2891**

✓ Sob settled here?

22/01/2020

**Souvik Ghosh**

Souvik Ghosh changed their phone number to a new n...

19/04/2020

**MakeMyTrip**

Dear Raja, We hope you had a pleasant journey

08/10/2020



*Figure 8: Step 2*

15:19



MakeMyTrip ✓

4 JANUARY



ETicket\_NF7913125617853:

PDF

8 JANUARY

Dear Raja,

View contact

Report

Block

Search

Mute notifications

Wallpaper

More ▶

We hope you had a pleasant journey from **Deini** to **Kolkata**.  
Considering your overall experience, how likely would you be to  
recommend MakeMyTrip to your friends and family?

Please rate us on a scale of 0 to 10 by clicking  
<https://bit.ly/2tK1dUz?open=outside>

17:56

11 NOVEMBER 2020

Your security code with MakeMyTrip changed. Tap to learn more.

This chat is with the official business account of MakeMyTrip. Tap to learn more.

5 DECEMBER 2020

Your security code with MakeMyTrip changed. Tap to learn more.

This chat is with the official business account of MakeMyTrip. Tap to learn more.

8 DECEMBER 2020

Your security code with MakeMyTrip changed. Tap to learn more.

This chat is with the official business account of MakeMyTrip. Tap to learn more.



Type a message



*Figure 9: Step 3*

15:19



MakeMyTrip ✓

4 JANUARY



Media, links, and docs

Clear chat

Export chat

Add shortcut



ETicket\_NF79131256178532.pdf

PDF

15:42

8 JANUARY 2020

Dear Raja,

We hope you had a pleasant journey from **Delhi** to **Kolkata**. Considering your overall experience, how likely would you be to recommend MakeMyTrip to your friends and family?

Please rate us on a scale of 0 to 10 by clicking  
<https://bit.ly/2tK1dUz?open=outside>

17:56

11 NOVEMBER 2020

Your security code with MakeMyTrip changed. Tap to learn more.

This chat is with the official business account of MakeMyTrip. Tap to learn more.

5 DECEMBER 2020

Your security code with MakeMyTrip changed. Tap to learn more.

This chat is with the official business account of MakeMyTrip. Tap to learn more.

8 DECEMBER 2020

Your security code with MakeMyTrip changed. Tap to learn more.

This chat is with the official business account of MakeMyTrip. Tap to learn more.



Type a message



*Figure 10: Step 4*

## The structure of WhatsApp chat data

Following is the list of fields in the data provided by WhatsApp:

|   |
|---|
| WhatsApp:   |
| WhatsApp: WhatsApp:                               |
| WhatsApp: WhatsApp: WhatsApp: WhatsApp:           |
| WhatsApp: WhatsApp: WhatsApp: WhatsApp: WhatsApp: |
| WhatsApp: WhatsApp: WhatsApp: WhatsApp: WhatsApp: |
| WhatsApp: WhatsApp: WhatsApp: WhatsApp: WhatsApp: |
| WhatsApp: WhatsApp: WhatsApp: WhatsApp:           |
| WhatsApp:   |
| WhatsApp:   |

|   |
|---|
| WhatsApp:                               |
| WhatsApp:                               |
| WhatsApp: WhatsApp: WhatsApp: WhatsApp: |

**Table 1:** Fields provided in WhatsApp chat data

## Reading WhatsApp chat data using R

One of the ways to read WhatsApp chat data using R is with the help of the library This library provides a function – rwa\_read() – to read the TXT file containing the WhatsApp chat data. The usage is shown as follows:

```
library(rwhatsapp)
v_Chats <- rwa_read("WhatsApp-MAHE-20201216.txt")
```

We can view the read data using the head() function:

```
head(v_Chats, 5)
##                      time          author
## 1 2020-10-17 09:55:39 PGAIDL-SEP INTAKE 2020
## 2 2020-10-17 09:55:39
## 3 2020-10-17 10:19:02
## 4 2020-10-17 10:33:34
## 5 2020-10-17 11:22:32
##
text
## 1 Messages and calls are end-to-end encrypted. No one
outside of this chat, not even WhatsApp, can read or listen to
them.
##
2
+971 58 178 6877 created this group
```

```
## 3
+971 58 178 6877 added you
## 4
+971 58 178 6877 changed this group's icon
## 5           Hi
##               source id emoji      emoji_name

## 1 WhatsApp-MAHE-20201216.txt 1 NULL      NULL
## 2 WhatsApp-MAHE-20201216.txt 2 NULL      NULL
## 3 WhatsApp-MAHE-20201216.txt 3 NULL      NULL
## 4 WhatsApp-MAHE-20201216.txt 4 NULL      NULL
## 5 WhatsApp-MAHE-20201216.txt 5 NULL      NULL
```

## Visualizing WhatsApp chat data

Once we have the WhatsApp chat data, we can create numerous visualizations. I will discuss a few and leave the rest for you to explore by yourself. While discussing these visualizations, I will discuss some features of the `ggplot2` library.

Before we create our first visualization, we will extract some information from the data. For this code to work, you would need to include the libraries `lubridate` and `dplyr` shown as follows:

```
v_Chats <- v_Chats %>%
  mutate(message_date = date(time)) %>%
  mutate(message_month = month(time, label = TRUE)) %>%
  mutate(message_month = factor(message_month)) %>%
  mutate(message_weekday_number = wday(message_date)) %>%
  mutate(message_weekday_name = weekdays(message_date)) %>%
  mutate(message_weekday_name = factor(message_weekday_name))
%>%
  mutate(message_hour = hour(time)) %>%
  filter(!is.na(author))
```

## Messages per day

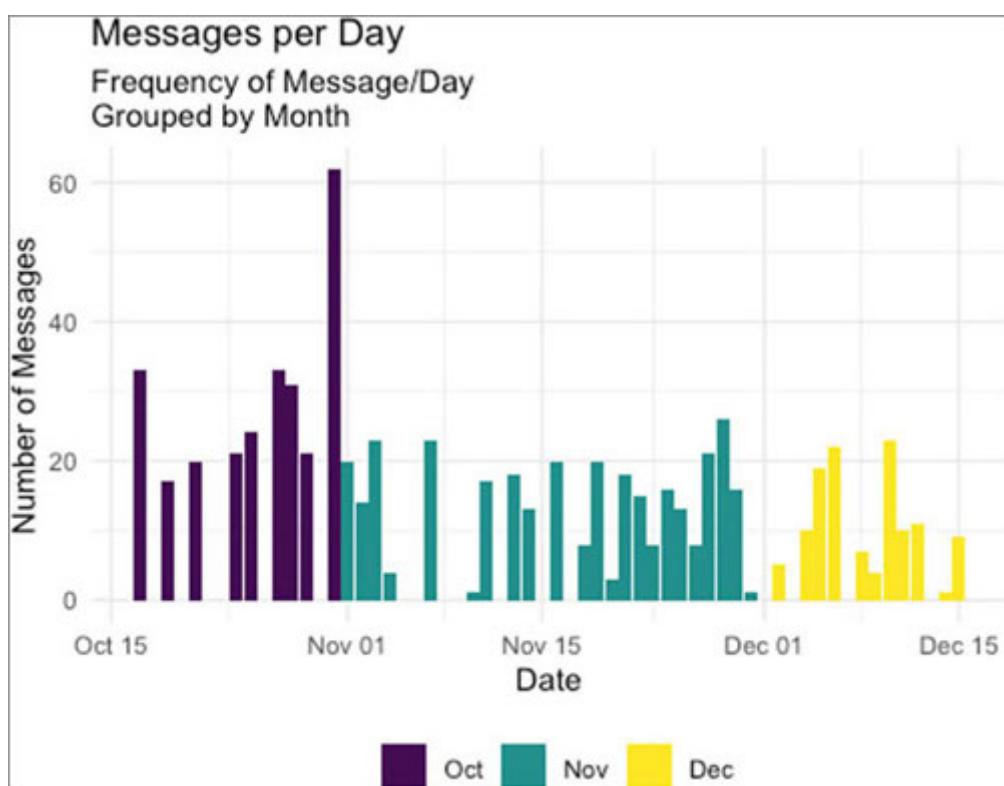
The first visualization we will discuss is to check the number of messages per day. The code is provided as follows. You should know that we would need the libraries and In addition to these, we would need the library viridis:

```
v_Chats %>%
  group_by(message_month) %>%
  count(message_date) %>%
  ggplot(aes(x = message_date, y = n, fill = message_month)) +
  geom_bar(stat = "identity") +
  scale_fill_viridis(discrete = TRUE) +
  labs(x = "Date", y = "Number of Messages", fill = "Month - ") +
  ggtitle("Messages per Day", "Frequency of Message/Day \nGrouped by Month") +
  theme_minimal() +
  theme(legend.title = element_blank(),
    legend.position = "bottom")
```

It should be clear from the code that we are plotting message\_date on the X-axis and the number of messages on the Y-axis. The fill parameter (that is, distinguishes each message\_date by its month. The option geom\_bar() creates the bar chart. We use the scale\_fill\_viridis() function to color each month's data in a different color is the color palette).

Lastly, check I have stated that there is no need for any legend title with the function Also, I stated that legends should be positioned at the bottom with the parameter

The output is shown as follows:



**Figure 11:** Visualization - Messages Per Day

We will now take a small digression and check how we can control the different aspects of this chart.

We can control the title and the subtitle using the `plot.title` and `plot.subtitle` parameters, respectively. We can control the axis titles using the parameter The axis text can be controlled using the parameters `axis.text.x` and `axis.text.y` for the X-axis and Y-axis,

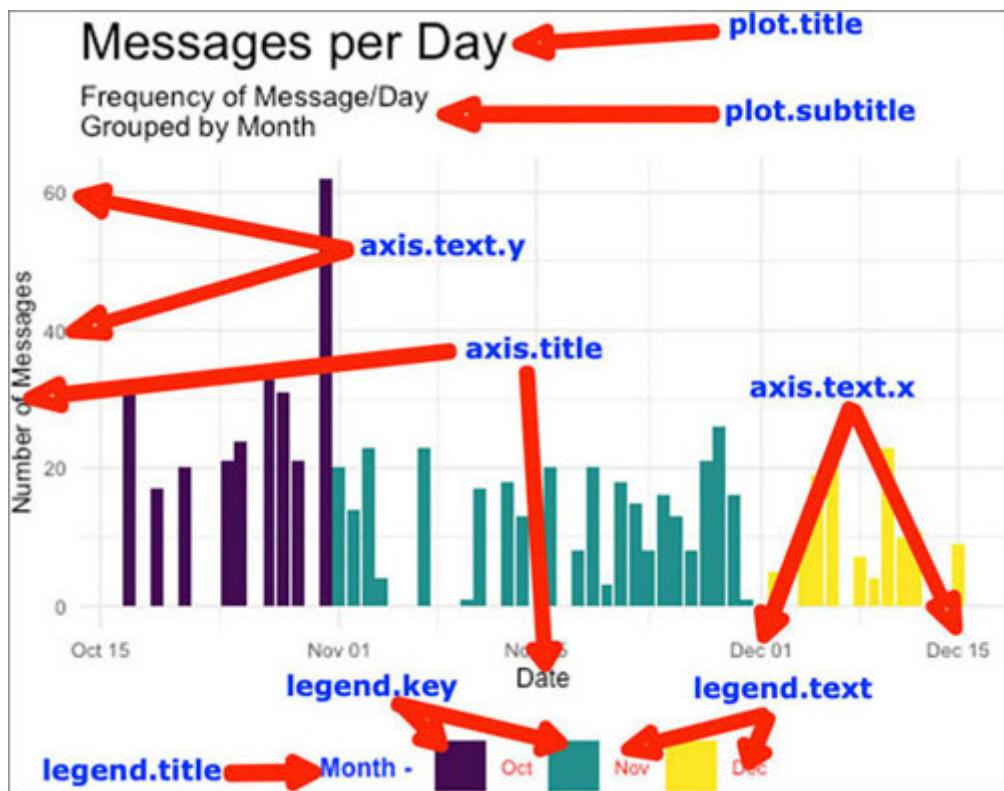
respectively. The Legend Title can be controlled by the parameter `legend.title`. The legends can be controlled using the `legend.text`, and `legend.key.width` parameters.

The function `element_text()` can be used to set the color, size, font face, and angle. The code is provided as follows:

```
v_Chats %>%
```

```
group_by(message_month) %>%
count(message_date) %>%
ggplot(aes(x = message_date, y = n, fill = message_month)) +
geom_bar(stat = "identity") +
scale_fill_viridis(discrete = TRUE) +
labs(x = "Date", y = "Number of Messages", fill = "Month - ") +
ggtitle("Messages per Day", "Frequency of Message/Day \nGrouped
by Month") +
theme_minimal() +
theme(legend.title = element_text(color = "blue", size = 9, face =
"bold"),
legend.text = element_text(color = "red", size = 7),
legend.position = "bottom",
legend.key.size = unit(0.7, "cm"),
legend.key.width = unit(0.7, "cm"),
axis.text.x = element_text(size = 7),
axis.text.y = element_text(size = 7),
axis.title = element_text(size = 9),
plot.title = element_text(size = 18),
plot.subtitle = element_text(size = 10))
```

The output is shown as follows:



*Figure 12: Parameters to control the chart*

## Messages per weekday.

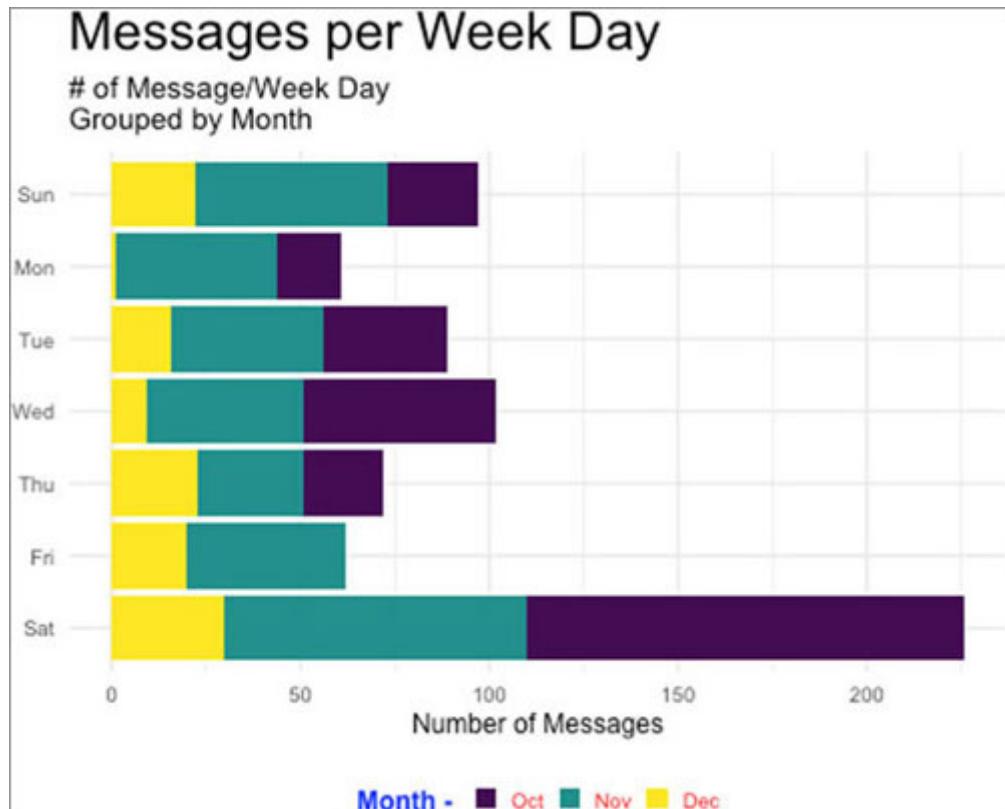
Using the following code, we can visualize the messages per weekday:

```
v_Chats %>%
  group_by(message_month, message_weekday_number,
  message_weekday_name) %>%
  count() %>%
  ggplot(aes(x = reorder(message_weekday_name, -
  message_weekday_number),
  y = n,
  fill = message_month)) +
  geom_bar(stat = "identity") +
  scale_x_discrete(labels = c("Monday" = "Mon",
  "Tuesday" = "Tue",
  "Wednesday" = "Wed",
  "Thursday" = "Thu",
  "Friday" = "Fri",
  "Saturday" = "Sat",
  "Sunday" = "Sun")) +
  scale_fill_viridis(discrete = TRUE) +
  labs(x = "", y = "Number of Messages", fill = "Month - ") +
  coord_flip() +
  ggtitle("Messages per Week Day", "# of Message/Week
  Day\nGrouped by Month")+
  theme_minimal() +
```

```
theme(legend.title = element_text(color = "blue", size = 9, face =  
"bold"),  
legend.text = element_text(color = "red", size = 7),  
legend.position = "bottom",  
  
legend.key.size = unit(0.3, "cm"),  
legend.key.width = unit(0.3, "cm"),  
axis.text.x = element_text(size = 7),  
axis.text.y = element_text(size = 7),  
axis.title = element_text(size = 9),  
plot.title = element_text(size = 18),  
plot.subtitle = element_text(size = 10))
```

Notice the use of the function `coord_flip()` to flip the axis. Also, notice that I used the `reorder()` function to order the weekdays by the weekday number. So, the weekdays appear in order.

The output is shown as follows:



*Figure 13: Visualization - Messages per Weekday*

## Radar chart

The same data regarding messages per weekday can be visualized as a radar chart. However, this requires some processing of the data. We will go through the process step-by-step.

First, we need to count the number of messages for every weekday for every month. We can get this done using the group\_by() function and the count() function from the dplyr library shown as follows:

```
v_Chats %>%
  group_by(message_month, message_weekday_number) %>%
  count()
## # A tibble: 20 x 3
## # Groups:   message_month, message_weekday_number [20]
##       message_month message_weekday_number     n
##                 <dbl>                  <dbl>
## 1          1 Oct                      1     24
## 2          2 Oct                      2     17
## 3          3 Oct                      3     33
## 4          4 Oct                      4     51
## 5          5 Oct                      5     21
## 6          6 Oct                     7    116
## 7          7 Nov                      1     51
## 8          8 Nov                      2     43
## 9          9 Nov                      3     40
## 10        10 Nov                     4     42
```

|           |   |    |
|-----------|---|----|
| ## 11 Nov | 5 | 28 |
| ## 12 Nov | 6 | 42 |
| ## 13 Nov | 7 | 80 |
| <br>      |   |    |
| ## 14 Dec | 1 | 22 |
| ## 15 Dec | 2 | 1  |
| ## 16 Dec | 3 | 16 |
| ## 17 Dec | 4 | 9  |
| ## 18 Dec | 5 | 23 |
| ## 19 Dec | 6 | 20 |
| ## 20 Dec | 7 | 30 |

However, we need this data tabulated such that each weekday is a separate column. This can be accomplished using the `spread()` function from the `dplyr` library shown as follows:

```
v_temp <- v_Chats %>%
  group_by(message_month, message_weekday_number) %>%
  count() %>%
  spread(message_weekday_number, n) %>%
  rename("Sunday" = "1",
         "Monday" = "2",
         "Tuesday" = "3",
         "Wednesday" = "4",
         "Thursday" = "5",
         "Friday" = "6",
         "Saturday" = "7")
v_temp
## # A tibble: 3 x 8
## # Groups:   message_month [3]
```

```

##    message_month Sunday Monday Tuesday Wednesday
Thursday Friday Saturday
##  

## 1 Oct          24      17      33      51
21      NA       116
## 2 Nov          51      43      40      42
28      42       80
## 3 Dec          22      1       16      9
23      20       30

```

For being able to create the radar chart, we need two more rows in the data frame v\_temp – one row for the maximum value for each column and one row for the minimum value for each column. We know that the minimum value must be ZERO. However, we need to determine the maximum value.

As we can see in the contents of there can be some values which are NA. This will cause a problem in determining the maximum value. So, we will convert all the NAs to ZERO shown as follows:

```
v_temp[is.na(v_temp)] <- 0  
v_temp  
## # A tibble: 3 x 8  
## # Groups:   message_month [3]  
##   message_month Sunday Monday Tuesday Wednesday  
##   Thursday Friday Saturday  
##  
## 1 Oct          24      17      33      51  
21          0       116
```

```

## 2 Nov          51    43    40    42
28     42      80

## 3 Dec          22     1    16     9
23     20      30

```

Now we can determine the maximum number of messages on a weekday shown as follows. Note that we use the function `sapply()` to consider only the numeric columns for determining the maximum number of messages:

```

v_maxValue <- max(v_temp[sapply(v_temp, is.numeric)])
v_maxValue
## [1] 116

```

What we would like is to round the maximum number of messages to the nearest This can be done using the `round_any()` function from the `plyr` library shown as follows:

```

library(plyr)
v_roundedMaxValue <- round_any(v_maxValue, 10, f = ceiling)
v_roundedMaxValue
## [1] 120

```

**When both the `plyr` and the `dplyr` libraries are required in a program, the `plyr` library must always be loaded before the `dplyr` library.**

Now, we can create a temporary data frame to contain the maximum value and the minimum value for each weekday. Then,

we can append this data frame to our main data frame - The maximum and the minimum values should be the first two rows of the data frame. The complete code is provided as follows:

```
v_min.MaxValue <- data.frame("Sunday" = c(v_rounded.MaxValue, o),  
  
"Monday" = c(v_rounded.MaxValue, o),  
"Tuesday" = c(v_rounded.MaxValue, o),  
"Wednesday" = c(v_rounded.MaxValue, o),  
"Thursday" = c(v_rounded.MaxValue, o),  
"Friday" = c(v_rounded.MaxValue, o),  
"Saturday" = c(v_rounded.MaxValue, o))  
rownames(v_min.MaxValue) <- c("Max", "Min")  
v_temp1 <- dplyr::bind_rows(v_min.MaxValue, v_temp)  
v_temp1$message_month <- as.character(v_temp1$message_month)  
v_temp1[1,]$message_month <- "Max"  
v_temp1[2,]$message_month <- "Min"  
v_temp1$message_month <- as.factor(v_temp1$message_month)  
rownames(v_temp1) <- v_temp1$message_month  
v_temp1 <- select(v_temp1, -message_month)  
v_temp1  
##      Sunday Monday Tuesday Wednesday Thursday Friday  
Saturday  
## Max     120     120     120  
120     120     120     120  
## Min      o      o      o  
o      o      o      o  
## Oct     24      17      33      51  
21      o     116  
## Nov     51      43      40      42      28  
42      80
```

```
## Dec      22      1      16      9      23
20      30
```

Now, the data frame v\_temp1 contains the data which is correct for creating the radar chart.

We have one other task before we can create the radar chart and that is to generate the axes labels. Now, we will generate five labels for the axes. We would like these labels to be equidistant. So, we can use the following code to create this sequence:

```
v_numberOfBins <- 4
v_stepValue <- round(v_rounded.MaxValue / v_numberOfBins, 0)
v_stepValue
## [1] 30
v_axeslabels <- seq(0, v_rounded.MaxValue, v_stepValue)
v_axeslabels
## [1] 0 30 60 90 120
```

One last task is to declare the colors for the different months. We know that there can be a maximum of 12 months and thus, we will declare 12 colors. I have declared 12 colors at random. You can change this as per your taste:

```
v_Colors <- c("cyan", "orange", "red",
"brown1", "bisque4", "coral3",
"cornflowerblue", "cyan4", "darkolivegreen3",
"darkorchid2", "gold3", "goldenrod3")
```

Now, we are ready to generate the radar chart. We will create the radar chart using the `radarchart()` function in the `fmsb` library shown as follows:

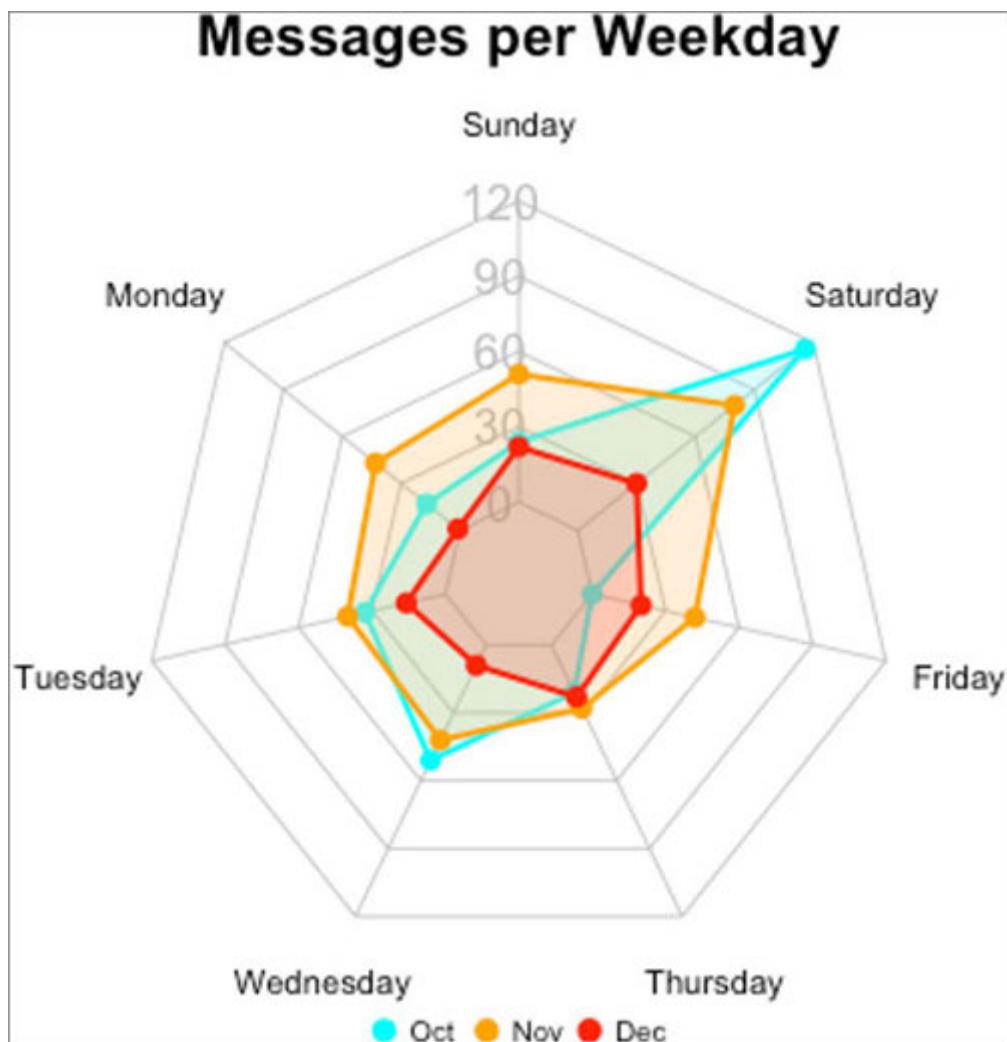
```
library(fmsb)
```

```
op <- par(mar = c(1, 2, 2, 2))
radarchart(
  v_temp1, axistype = 1,
  pcol = v_Colors, pfcol = scales::alpha(v_Colors, 0.2), plwd = 2,
  plty = 1,
  cglcol = "grey", cglty = 1, cglwd = 0.8,
  axislabcol = "grey",
  vlcex = 0.7, vlables = colnames(v_temp1),
  caxislabels = v_axeslabels,
  title = "Messages per Weekday"
)
legend(
  x = "bottom", legend = rownames(v_temp1[-c(1,2),]), horiz = TRUE,
  bty = "n", pch = 20,
  col = v_Colors,
  text.col = "black", cex = .6, pt.cex = 1.5
)
par(op)
```

Note that in the legends, we remove the first two rows of the data frame. This is because the `radarchart()` function uses rows containing the maximum and the minimum values internally and is not a part of the output.

I leave it to you to explore all the parameters of the `radarchart()` function.

The output is shown as follows:



*Figure 14: Radar chart*

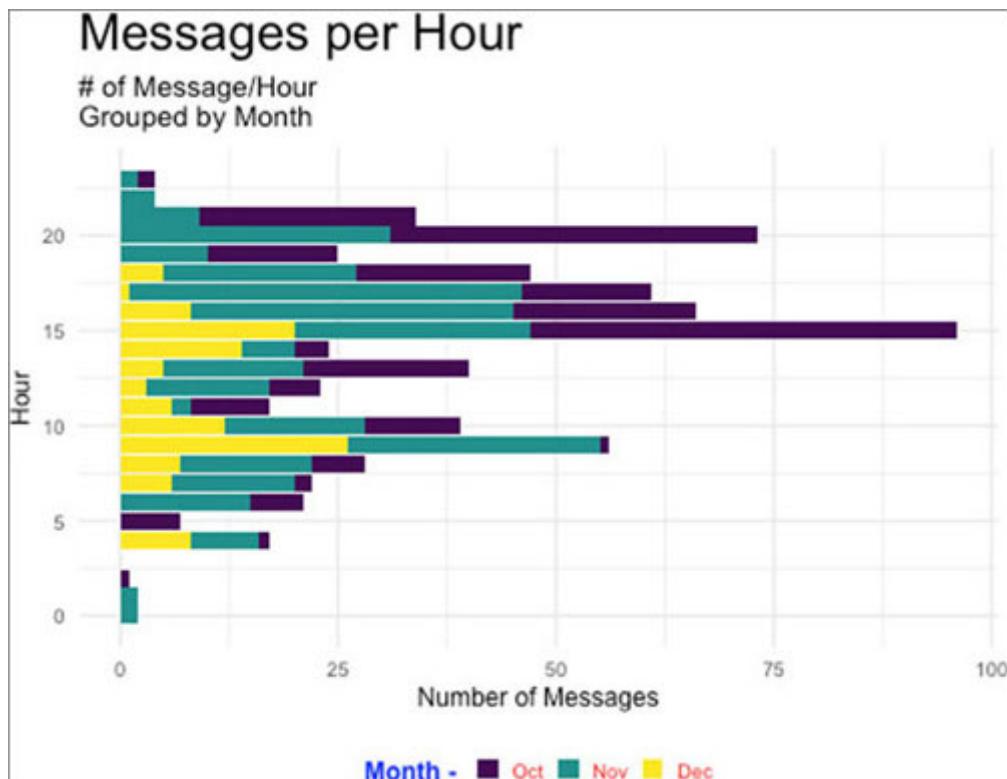
## Messages per hour

We can visualize the messages per hour just like how we visualized the messages per weekday. Here, we group the data by the month and hour. The code is provided as follows:

```
v_Chats %>%
  group_by(message_month, message_hour) %>%
  count() %>%
  ggplot(aes(x = message_hour,
             y = n,
             fill = message_month)) +
  geom_bar(stat = "identity") +
  scale_fill_viridis(discrete = TRUE) +
  labs(x = "Hour", y = "Number of Messages", fill = "Month - ") +
  coord_flip() +
  ggtitle("Messages per Hour", "# of Message/Hour \nGrouped by Month") +
  theme_minimal() +
  theme(legend.title = element_text(color = "blue", size = 9, face =
  "bold"),
        legend.text = element_text(color = "red", size = 7),
        legend.position = "bottom",
        legend.key.size = unit(0.3, "cm"),
        legend.key.width = unit(0.3, "cm"),
        axis.text.x = element_text(size = 7),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 9),
```

```
plot.title = element_text(size = 18),  
plot.subtitle = element_text(size = 10))
```

The output is shown as follows:



**Figure 15:** Visualization - Messages per Hour

We could fine grain this visualization by adding another dimension of the weekday. We will see how the messages were exchanged in every month for every weekday in every hour.

We could add the dimension of weekday by using the faceting feature of `ggplot2` which allows for creating multi-panel graphs. The code is provided as follows:

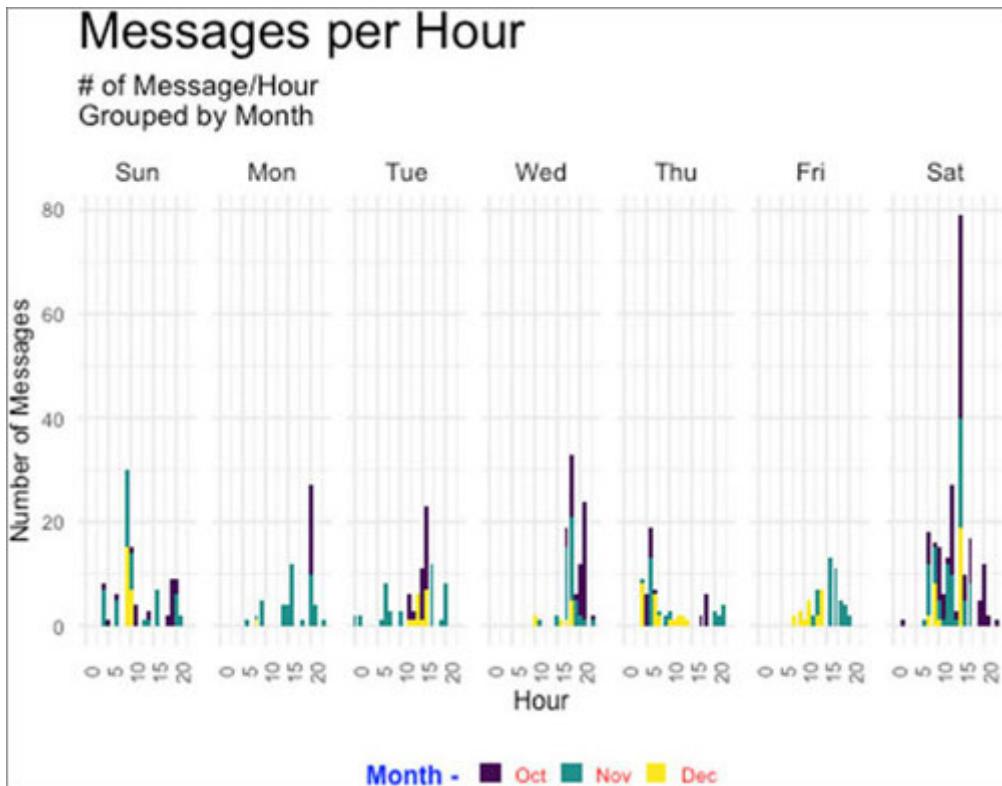
```
v_weekdays <- c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
names(v_weekdays) <- 1:7
v_Chats %>%
  group_by(message_month, message_weekday_number,
  message_weekday_name, message_hour) %>%
  
  count() %>%
  ggplot(aes(x = message_hour,
y = n,
fill = message_month)) +
  geom_bar(stat = "identity") +
  scale_fill_viridis(discrete = TRUE) +
  labs(x = "Hour", y = "Number of Messages", fill = "Month - ") +
  facet_wrap(~message_weekday_number, ncol=7,
labeller = labeller(message_weekday_number = v_weekdays)) +
  ggtitle("Messages per Hour", "# of Message/Hour \nGrouped by
Month") +
  theme_minimal() +
  theme(legend.title = element_text(color = "blue", size = 9, face =
"bold"),
legend.text = element_text(color = "red", size = 7),
legend.position = "bottom",
legend.key.size = unit(0.3, "cm"),
legend.key.width = unit(0.3, "cm"),
axis.text.x = element_text(size = 7, angle = 90),
axis.text.y = element_text(size = 7),
axis.title = element_text(size = 9),
plot.title = element_text(size = 18),
plot.subtitle = element_text(size = 10))
```

Notice that now we have grouped the data based on message month, weekday number, weekday name, and message hour. I included both weekday number and weekday name because i want to order according to the weekday number and display the weekday name.

Notice that in the function we create the facet according to the weekday number and we use the labeller parameter to show the weekday names in the graph. I created a variable named v\_weekdays to store the weekday names as they should be displayed and named the values as per the weekday numbers. So, the labeller() function picks up the weekday names as per the weekday number.

Also, notice the parameter axis.text.x in the theme() function. We have used the angle parameter to display the X-axis labels at an angle of 90 degrees.

The output is shown as follows:



**Figure 16:** Visualization - Messages Per Hour using facet on Weekday

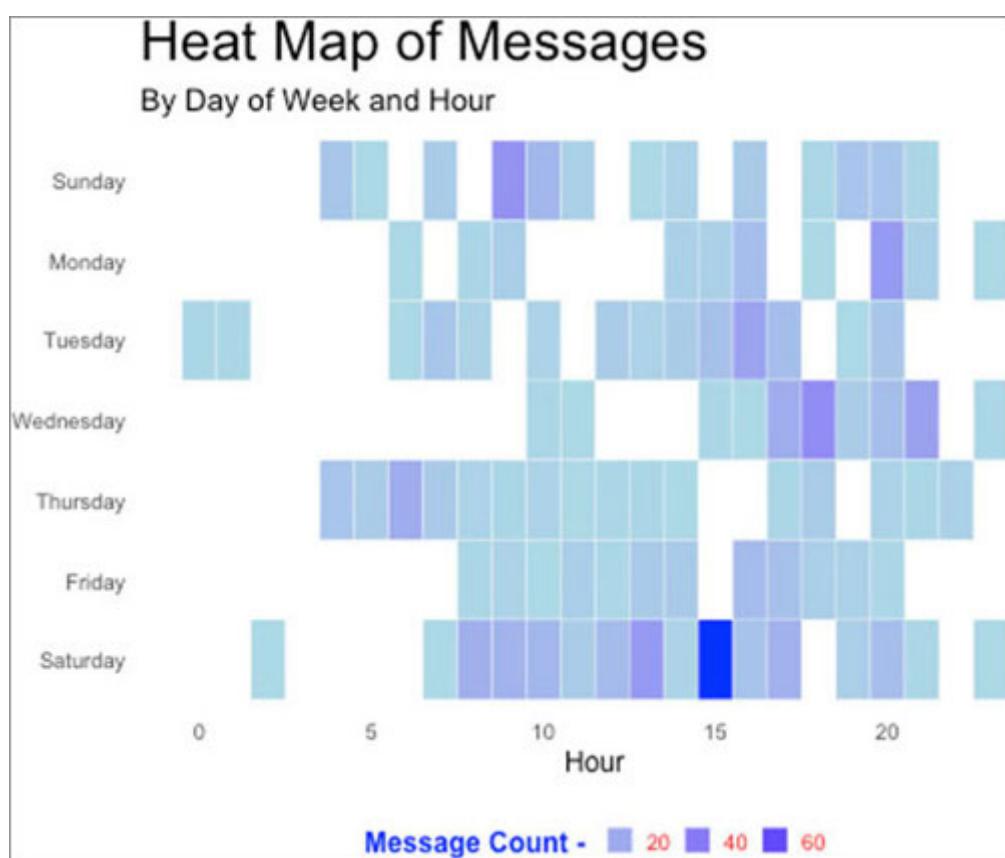
## Heat map

We can visualize this data regarding messages per hour in the form of a heat map. The code to generate the heat map for this data is provided as follows:

```
v_Chats %>%
  group_by(message_weekday_number, message_weekday_name,
  message_hour) %>%
  count() %>%
  ggplot(aes(x = message_hour,
  y = reorder(message_weekday_name, -message_weekday_number)))
  +
  geom_tile(aes(fill = n), color = "white", na.rm = TRUE) +
  scale_fill_gradient(low = "lightblue", high = "blue") +
  guides(fill = guide_legend(title = "Message Count - ")) +
  labs(title = "Heat Map of Messages", subtitle = "By Day of Week
and Hour",
  x = "Hour", y = "") +
  theme_minimal() +
  theme(legend.title = element_text(color = "blue", size = 9, face =
  "bold"),
  legend.text = element_text(color = "red", size = 7),
  legend.position = "bottom",
  legend.key.size = unit(0.3, "cm"),
  legend.key.width = unit(0.3, "cm"),
  axis.text.x = element_text(size = 7),
  axis.text.y = element_text(size = 7),
```

```
axis.title = element_text(size = 9),  
plot.title = element_text(size = 18),  
plot.subtitle = element_text(size = 10),  
  
panel.grid.major = element_blank(),  
panel.grid.minor = element_blank())
```

The output is shown as follows:



*Figure 17: Heat map*

## Messages per author

We can create many visualizations for messages per author. I will create just one visualization which will display the number of messages per author per weekday across all the months.

The code is provided as follows:

```
v_Chats %>%
  group_by(message_month, message_weekday_number,
  message_weekday_name, author) %>%
  count() %>%
  ggplot(aes(x = reorder(author, n),
  y = n,
  fill = reorder(message_weekday_name, message_weekday_number)))
  +
  geom_bar(stat = "identity") +
  scale_fill_manual(labels = c("Monday" = "Mon",
  "Tuesday" = "Tue",
  "Wednesday" = "Wed",
  "Thursday" = "Thu",
  "Friday" = "Fri",
  "Saturday" = "Sat",
  "Sunday" = "Sun"),
  values = brewer.pal(8, "Dark2")) +
  facet_wrap(~message_month) +
  labs(x = "", y = "Number of Messages", fill = "Week Day - ") +
  coord_flip()
```

```
ggtitle("Messages per Author", "Number of Messages/Author  
\nGrouped by Week Day") +  
theme_minimal() +  
  
theme(legend.title = element_text(color = "blue", size = 9, face =  
"bold"),  
legend.text = element_text(color = "red", size = 7),  
legend.position = "bottom",  
legend.key.size = unit(0.3, "cm"),  
legend.key.width = unit(0.3, "cm"),  
axis.text.x = element_text(size = 7),  
axis.text.y = element_text(size = 5),  
axis.title = element_text(size = 9),  
plot.title = element_text(size = 18),  
plot.subtitle = element_text(size = 10))
```

Notice that we faceted on message month. So, we get a panel for each month. Under each month, we can visualize the number of messages per weekday.

The output is shown as follows:



**Figure 18:** Visualization - Messages per Author

We can create a heat map of message per author per weekday. The code is provided as follows:

```
v_Chats %>%
  group_by(message_weekday_number, message_weekday_name,
  author) %>%
  count() %>%
  ggplot(aes(x = author,
  y = reorder(message_weekday_name, -message_weekday_number)))
 +
  geom_tile(aes(fill = n), color = "white", na.rm = TRUE) +
  scale_fill_gradient(low = "lightblue", high = "blue") +
  scale_y_discrete(labels = c("Monday" = "Mon",
```

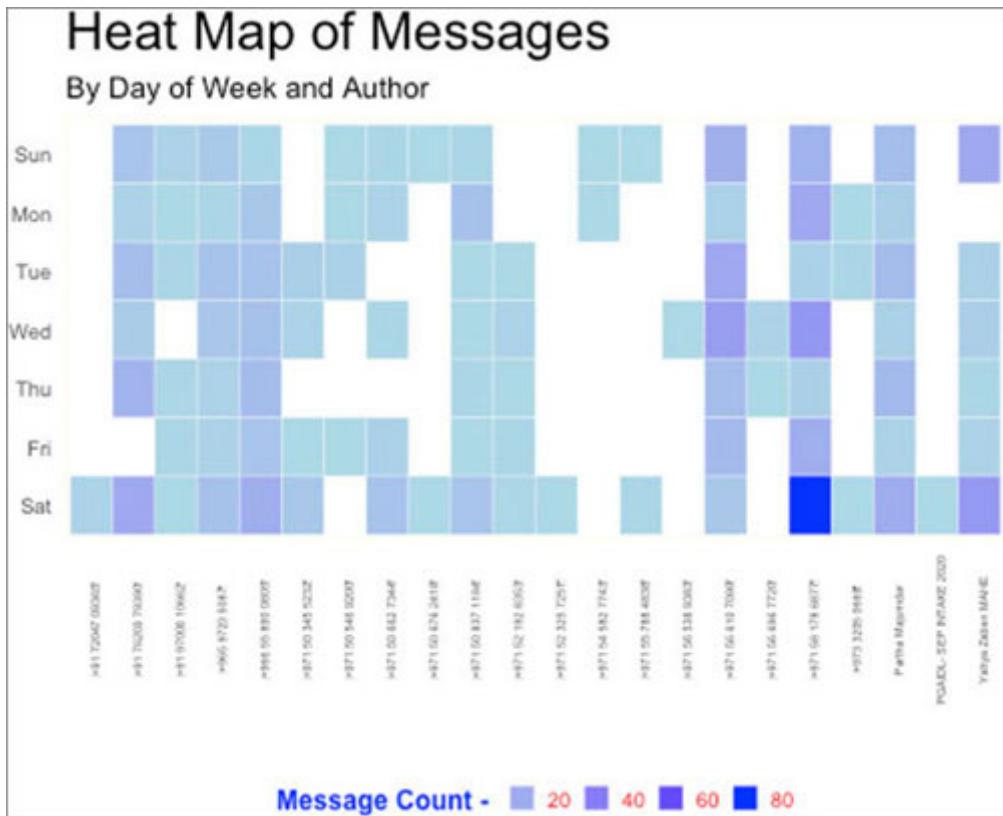
```

"Tuesday" = "Tue",
"Wednesday" = "Wed",
"Thursday" = "Thu",
"Friday" = "Fri",
"Saturday" = "Sat",
"Sunday" = "Sun")) +
guides(fill = guide_legend(title = "Message Count - ")) +
labs(title = "Heat Map of Messages", subtitle = "By Day of Week
and Author",
x = "", y = "") +
theme_minimal() +
theme(legend.title = element_text(color = "blue", size = 9, face =
"bold"),
legend.text = element_text(color = "red", size = 7),
legend.position = "bottom",

legend.key.size = unit(0.3, "cm"),
legend.key.width = unit(0.3, "cm"),
axis.text.x = element_text(size = 4, angle = 90),
axis.text.y = element_text(size = 7),
axis.title = element_text(size = 9),
plot.title = element_text(size = 18),
plot.subtitle = element_text(size = 10),
panel.grid.major = element_blank(),
panel.grid.minor = element_blank(),
panel.background = element_rect(color = "lightyellow"))

```

The output is shown as follows:



**Figure 19:** Heat map of messages per author per weekday

## Emoji analysis

WhatsApp chat data provides two fields with information on the Emojis used in the chats. These fields are emoji and So, a lot of analysis is possible on the Emojis used in the chats. I will discuss two such visualizations and leave it your creativity to conduct many other. Before we can create visualizations on Emojis, we need to prepare some data. The main purpose of this data preparation is to get the images of the Emojis as the Emoji codes are stored in WhatsApp chat data.

We start by checking the actual data provided by WhatsApp:

```
head(v_Chats[,c("emoji", "emoji_name")]) %>%
filter(!is.null(emoji_name)) %>%
filter(emoji_name != "NULL")
## moji
## 1 \U0001f64f
## 2 \U0001f6oa
## 3 \U0001f44d
## 4 \U0001f6oa, \U0001f6oa, \U0001f44d, \U0001f44d,
\U0001f44d
##
```

```

😊
## 6
\U0001f44d
##                                     emoji_name

## 1                               folded hands
## 2                               smiling face
with smiling eyes
## 3                               thumbs up
## 4 smiling face with smiling eyes, smiling face with smiling
eyes, thumbs up, thumbs up, thumbs up
## 5                               smiling face
## 6                               thumbs up

```

We clean the Emoji data and generate a URL from where we can get the image associated with the Emoji. The following code should be understandable as we have discussed all the functions used in this code earlier in the book:

```

v_Emojis <- v_Chats %>%
unnest(c(emoji, emoji_name)) %>%
mutate(emoji = str_sub(emoji, end = 1)) %>%
mutate(emoji_name = str_remove(emoji_name, ":.*")) %>%
mutate(emoji_url = map_chr(emoji,
~pasteo("https://abs.twimg.com/emoji/v2/72x72/",
as.hexmode(utf8ToInt(.x)), ".png")))
) %>%
filter(!is.null(emoji_name)) %>%
filter(emoji_name != "NULL")
head(v_Emojis)
## # A tibble: 6 x 13

```

```

##   time           author text  source      id emoji
emoji_name message_date
##
## 1 2020-10-17 17:38:38
## 2 2020-10-17 17:47:28
## 3 2020-10-21 20:41:08
## 4 2020-10-21 20:41:45
## 5 2020-10-21 20:41:45
## 6 2020-10-21 20:41:45
## # ... with 5 more variables: message_month ,
message_weekday_number ,
## #   message_weekday_name , message_hour , emoji_url

```

We can check the Emoji URLs generated as follows:

```

head(v_Emojis$emoji_url)
## [1] "https://abs.twimg.com/emoji/v2/72x72/1f64f.png"
## [2] "https://abs.twimg.com/emoji/v2/72x72/1f60a.png"
## [3] "https://abs.twimg.com/emoji/v2/72x72/1f44d.png"
## [4] "https://abs.twimg.com/emoji/v2/72x72/1f60a.png"
## [5] "https://abs.twimg.com/emoji/v2/72x72/1f60a.png"
## [6] "https://abs.twimg.com/emoji/v2/72x72/1f44d.png"

```

Now, we visualize the number of times each Emoji is used in the chat using the code shown as follows:

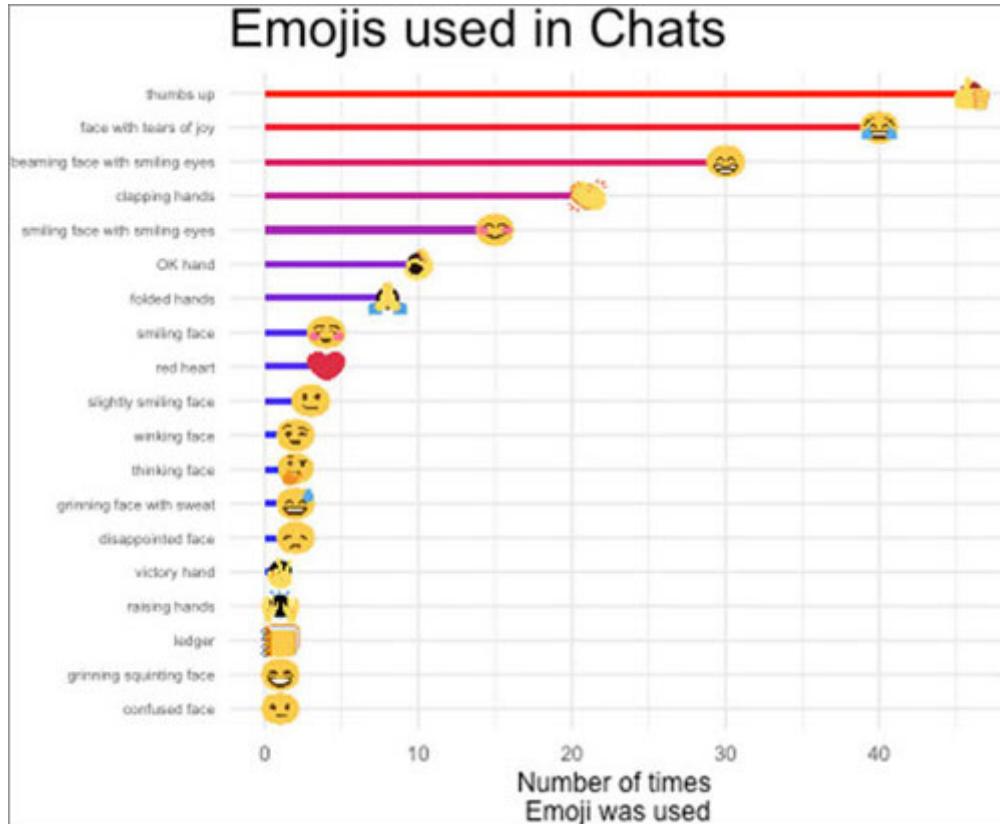
```

v_Emojis %>%
count(emoji, emoji_name, emoji_url) %>%
arrange(desc(n)) %>%

```

```
ggplot(aes(x = reorder(emoji_name, n), y = n)) +
  geom_col(aes(fill = n), show.legend = FALSE, width = .2) +
  geom_point(aes(color = n), show.legend = FALSE, size = 3) +
  geom_image(aes(image = emoji_url), size = .05) +
  scale_fill_gradient(low = "blue", high = "red") +
  scale_color_gradient(low = "black", high = "brown") +
  coord_flip() +
  ggtitle("Emojis used in Chats") +
  labs(x = "", y = "Number of times \nEmoji was used") +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 7),
        axis.text.y = element_text(size = 5),
        axis.title = element_text(size = 9),
        plot.title = element_text(size = 18),
        plot.subtitle = element_text(size = 10))
```

The output is shown as follows:



**Figure 20:** Visualization – The frequency of the usage of Emojis

Next, we create the visualization for the frequency of the usage of the Emojis by each author. The code is provided as follows:

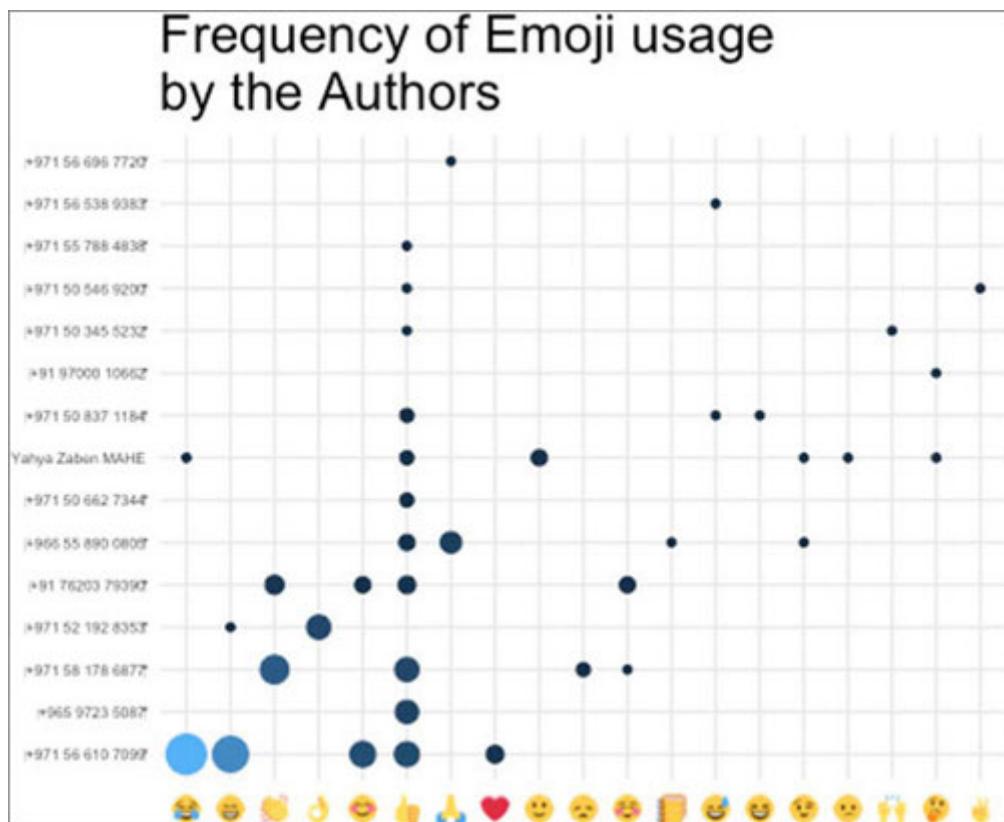
```
v_labels <- setNames(
pasteo("src='', v_Emojis$emoji_url, '' width='10' />"),
v_Emojis$emoji_url
)
v_Emojis %>%
group_by(author) %>%
count(author, emoji, emoji_name, emoji_url) %>%
arrange(desc(n)) %>%
ggplot(aes(x = reorder(emoji_url, -n), y = reorder(author, -n), fill =
n)) +
```

```

geom_point(aes(color = n, size = n), show.legend = FALSE) +
scale_x_discrete(name = NULL, labels = v_labels) +
ggtitle("Frequency of Emoji usage \nby the Authors") +
labs(x = "", y = "") +
theme_minimal() +
theme(axis.text.x = element_markdown(color = "black", size = 10),
axis.text.y = element_text(size = 5),
axis.title = element_text(size = 9),
plot.title = element_text(size = 18),
plot.subtitle = element_text(size = 10))

```

The output is shown as follows:



**Figure 21:** Visualization - Emoji usage by authors

## Word analysis

We round up our discussions on visualizing WhatsApp chat data by creating visualizations to analyze the words used in the messages. The text used in the messages is available in the column text. We start by removing all the emoticons, numbers, and URLs from the text shown as follows:

```
v_Chats$text1 <- gsub("[^\x01-\x7F]", "", v_Chats$text)
v_Chats <- v_Chats %>%
  mutate(tidy_text = gsub("\d", "", text1)) %>%
  mutate(tidy_text = gsub("(f|ht)tp(s?)://\\S+", "", tidy_text, perl=T))
```

Now, we visualize the top 30 words used in the messages using the following code:

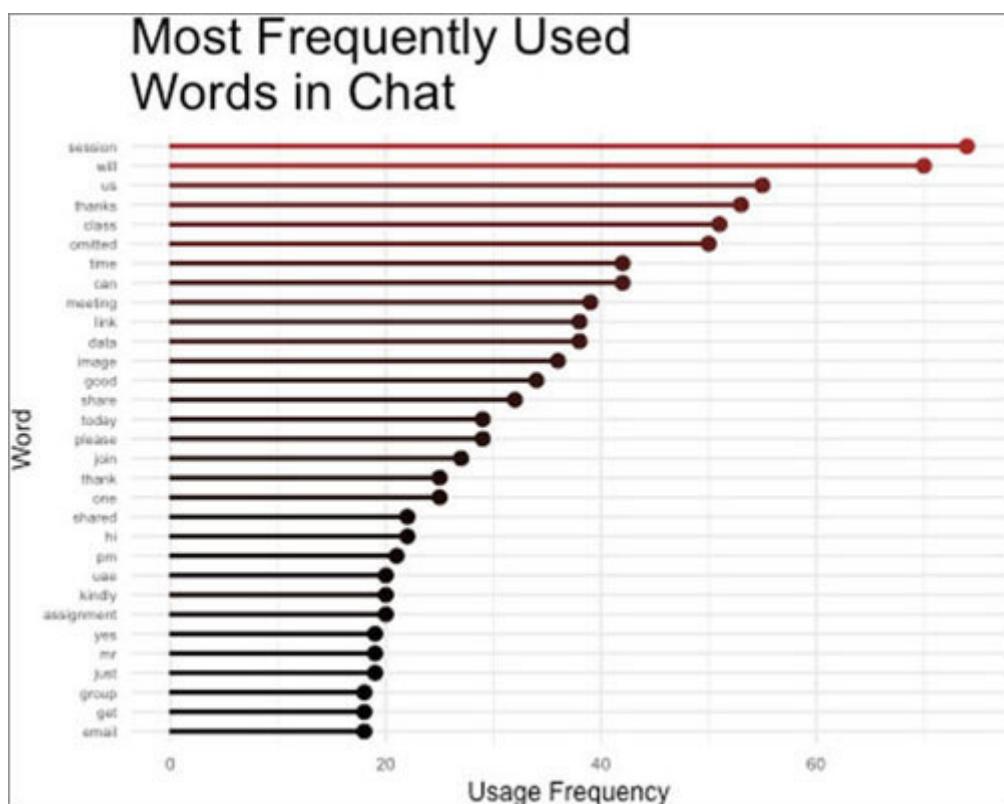
```
v_Chats %>%
  unnest_tokens(output = word, input = tidy_text) %>%
  filter(!word %in% tm::stopwords("english")) %>%
  count(word) %>%
  top_n(30, n) %>%
  arrange(desc(n)) %>%
  ggplot(aes(x = reorder(word, n), y = n, fill = n, color = n)) +
  geom_col(show.legend = FALSE, width = .1) +
  geom_point(show.legend = FALSE, size = 2) +
  scale_fill_gradient(low = "blue", high = "red") +
  scale_color_gradient(low = "black", high = "brown") +
```

```

ggtitle("Most Frequently Used \nWords in Chat") +
  labs(x = "Word", y = "Usage Frequency") +
  coord_flip() +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 6),
        axis.text.y = element_text(size = 5),
        axis.title = element_text(size = 9),
        plot.title = element_text(size = 18),
        plot.subtitle = element_text(size = 10))

```

Notice that we filtered out all the stop words using the `stopwords()` function from the `tm` library. The output is shown as follows:



**Figure 22:** Visualization - Word Frequency

Lastly, we create visualization for the frequency of the usage of the top words by all the authors. The code is provided as follows:

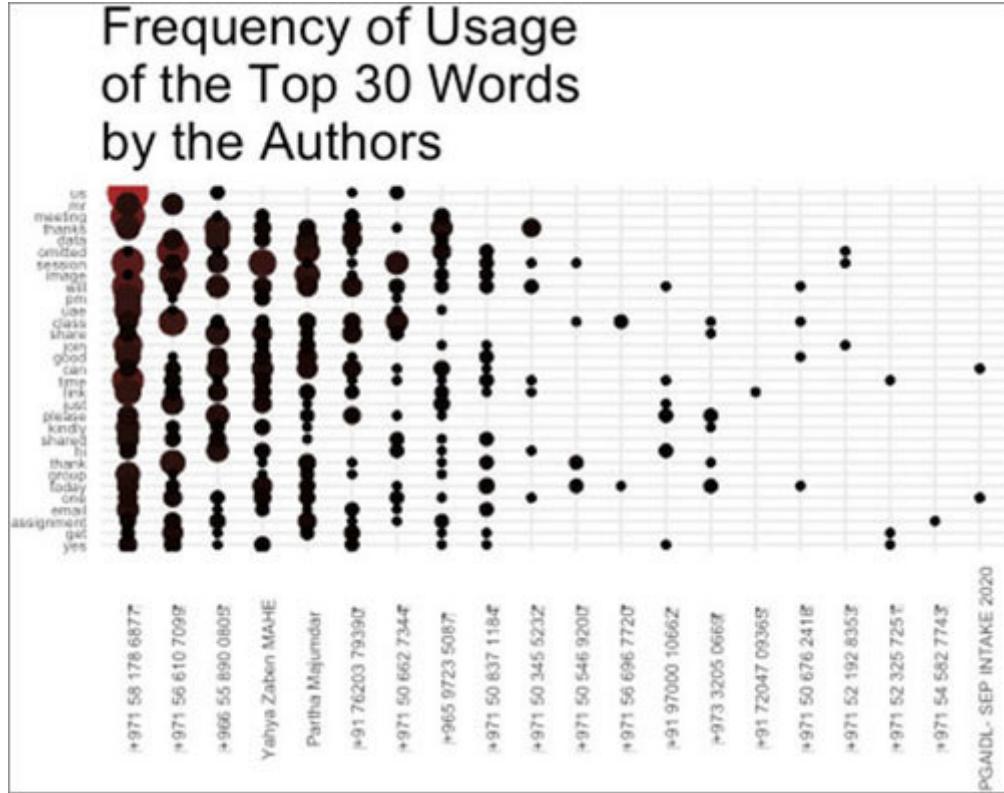
```
v_numberOfWords <- 30

v_topWords <- v_Chats %>%
  unnest_tokens(output = word, input = tidy_text) %>%
  filter(!word %in% tm::stopwords("english")) %>%
  count(word) %>%
  top_n(v_numberOfWords, n)

v_Chats %>%
  unnest_tokens(output = word, input = tidy_text) %>%
  filter(word %in% v_topWords$word) %>%
  group_by(author, word) %>%
  count(word) %>%
  arrange(desc(n)) %>%
  ggplot(aes(x = reorder(substring(word, 1,10), n), y = reorder(author,
  -n), fill = n, color = n)) +
  geom_point(aes(color = n, size = n), show.legend = FALSE) +
  scale_fill_gradient(low = "blue", high = "red") +
  scale_color_gradient(low = "black", high = "brown") +
  ggttitle(paste("Frequency of Usage \nof the Top", v_numberOfWords,
  "Words \nby the Authors", sep = " ")) +
  labs(x = "", y = "") +
  coord_flip() +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 6, angle = 90),
  axis.text.y = element_text(size = 5),
  axis.title = element_text(size = 9),
```

```
plot.title = element_text(size = 18),  
plot.subtitle = element_text(size = 10))
```

The output is shown as follows:



**Figure 23:** Visualization - Word Usage Frequency by Authors

## Conclusion

The analysis of WhatsApp data provides a wealth of information. There are hardly any products available in the world as on date (end of 2020) for analysis of WhatsApp data. So, this is an area where useful products could be created, and businesses could be formed around the same.

We began the chapter by discussing how to download data from WhatsApp. WhatsApp data can only be downloaded from mobile phones. It is not possible to download WhatsApp data from the desktop version of WhatsApp.

Then we discussed how to read WhatsApp chat data using R. After reading the WhatsApp chat data, we learned how to create various visualization of the WhatsApp chat data. During these discussions, we found out some more aspects of the ggplot2 library. We also discussed how to create two new types of charts – radar charts and heat maps.

In conclusion, I leave it to you to conduct Emotion Analysis on this data using the concepts discussed in this book.

### Points to remember

WhatsApp chat data can be read using the rwa\_read() function from the rwhatsapp library.

The round\_any() function from the plyr library can be used to round numbers for various requirements.

The seq() function can be used to create sequences.

The radarchart() function from the fmsb library can be used for creating radar charts. There are certain data preparation steps involved in creating radar charts using the radarchart() function.

Using the facet feature of the ggplot2 library, we can create multiple panels of visualization in a single chart.

### Multiple choice questions

The data provided by WhatsApp contains how many fields?

A. 20

B. 7

C. 12

D. None of these

To leave the legend title to be blank, which of these functions can we use?

A. element\_blank()

B. blank\_title() C. null\_title()

D. None of these

To adjust the Subtitle of the plot created using the ggplot() function, which of these parameters in the theme() function needs to be adjusted?

`graph.subtitle`

`subtitle`

`plot.subtitle`

None of these

Which of these functions can be used to rename columns in a data frame using the `dplyr` library?

`rename()`

`change_column_name()`

`rename_columns_name()`

None of these

## Answers to MCQs

B

A

C

A

## Questions

Conduct emotion analysis on the text in the messages.

Conduct emotion analysis on the Emojis in the messages.

Create a Shiny App for the analysis of WhatsApp chat data.

## Key terms

**SMS:** SMS is the acronym for Short Messaging Service. It was a very popular service when mobile phones were launched as users could exchange text messages between each other. However, there was a limitation that each text message could contain a maximum of 160 characters. Nevertheless, this is still a very economical way for exchanging messages and very widely used in the world.

## Index

### A

afinn lexicon [203](#)  
Android phones  
WhatsApp data, downloading from [492](#)  
Apple iPhones  
WhatsApp data, downloading from  
app.R [180](#)  
approaches, to sentiment analysis  
hybrid approach [200](#)  
knowledge-based approach [199](#)  
statistical approach [199](#)

### B

Bigram analysis  
about [248](#)  
process  
bing lexicon [204](#)  
Box and Whisker chart  
about [48](#)  
creating [48](#)

### C

Character User Interface (CUI) [113](#)

Chidiya  
client-side code [428](#)  
complete code [482](#)  
features [394](#)  
global code [399](#)  
landing page [394](#)

landing page, analyzing  
server-side code [417](#)  
URL [393](#)  
client-side code, Chidiya [428](#)  
code, ShabdhMegh  
app.R [180](#)  
ShabdhMegh.RMD  
code, ZEUSg  
app.R [273](#)  
ZEUSg.Rmd [321](#)  
Command Line Interface (CLI) [16](#)  
comparison cloud  
generating [219](#)  
Comprehensive R Archive Network (CRAN) reference link, for  
documentation [2](#)  
URL [2](#)  
convert2DigitNumbers() function [76](#)  
convert3DigitNumbers() function [77](#)  
convertCrores() function [80](#)  
convertLakhs() function [79](#)  
convertThousands() function [78](#)  
Corpora [91](#)  
Corpus [91](#)  
country analysis, Twitter data analysis  
conducting

CSV file  
data, reading from

## D

data

reading from CSV file  
reading from database  
data fetching, from Twitter  
fetched tweets, viewing  
rtweet library [345](#)  
tweets, fetching by search string [349](#).  
tweets, fetching for location  
Twitter account, authenticating [348](#)  
Twitter account, setting up  
data frame  
about [26](#)  
columns, adding  
columns, renaming [34](#)  
creating, from vectors  
data visualizations [47](#).  
empty data frame, creating  
rows, adding  
rows and columns, referencing  
rows based on conditions, referencing  
sorting [46](#)  
data from Twitter, analyzing  
about [364](#)  
country analysis, conducting  
hashtag analysis, conducting [37.7](#).

interactive maps, creating with leaflet() function [380](#)  
language analysis, conducting  
location analysis with maps, conducting  
place analysis, conducting [370](#)  
source analysis, conducting [375](#)

Times Series Chart, creating for tweets [365](#)  
user analysis, conducting  
data visualizations, data frame  
Box and Whisker chart [48](#)  
creating [47](#).  
histogram [47](#).  
pie chart [49](#).  
scatter plots [50](#)  
download button, ShabdhMegh client-side, programming [177](#).  
programming [177](#).  
RMarkdown, programming [180](#)  
server-side, programming [179](#).

## E

Emotion AI [198](#)  
emotion analysis [198](#)  
emotion analysis, conducting  
about [224](#)  
data, cleaning  
data, reading  
most prevalent emotion, determining in text  
sentiment flow in text  
stop words, removing [227](#).  
word analysis [228](#)

emotion analysis, of tweets  
conducting  
english() function [86](#)  
Extensible Messaging and Presence Protocol (XMPP) [487](#)

## F

fluid page layout [122](#)  
functions, Fetch button  
extractWords [420](#)  
refreshDisplay [420](#)  
saveStatistics [420](#)  
v\_tweets [420](#)  
functions, in R

## G

global code, Chidiya  
about [399](#)  
global functions [400](#)  
initialization code [412](#)  
required libraries, loading [400](#)  
global functions, Chidiya  
for extracting emotions from tweets [409](#)  
for extracting words from tweets [407](#)  
for fetching Previously Searched Subjects [404](#)  
for fetching subjects of tweets from database  
for reading initial set of tweets  
for saving statistics  
generic function, for displaying message box [401](#)

Google Maps API key  
about [337](#).  
credentials page [340](#)  
dashboard page [339](#).  
Google Maps menu [339](#).  
home page [337](#).  
home page, with logged in account [338](#)  
login page [338](#)

Sign In page [338](#)  
URL, for creating [337](#).  
Graphical User Interface (GUI) [113](#)

## H

hashtag analysis, Twitter data analysis  
conducting [377](#).  
helper() function  
histogram  
about [47](#).  
creating [47](#).  
hybrid approach, of sentiment analysis [200](#)

## I

initialization code, Chidiya  
about [412](#).  
database, setting up [414](#).  
Google Maps API, setting up [414](#).  
initial display, setting up  
lexicon, setting up [414](#).

system parameters, reading [412](#)  
Twitter Account, setting up [412](#)  
inputId [105](#)  
installation  
R software [5](#)  
Integrated Development Environment (IDE) [11](#)

## K

knowledge-based approach, of sentiment analysis [199](#)  
knowledge bases, for sentiment analysis [200](#)

## L

language analysis, Twitter data analysis  
conducting  
lexicons [202](#)  
lexicons, in R  
afinn lexicon [203](#)  
bing lexicon [204](#)  
loughran lexicon [205](#)  
nrc lexicon [206](#)  
libraries  
loading [18](#)  
library() command [18](#)  
library plotOutput() function [106](#)  
location analysis with maps, Twitter data analysis  
conducting  
loughran lexicon [205](#)

## N

Natural Language Processing (NLP) [325](#)

nrc lexicon [206](#)

number2wordsIndia() function [86](#)

number to words converter application

convert2DigitNumbers() function [76](#)

convert3DigitNumbers() function

convertCrores() function [80](#)

convertLakhs() function [79](#)

convertThousands() function [78](#)

creating [69](#)

function, writing

helper() function

number2wordsIndia() function [86](#)

R library [86](#)

trim() function [72](#)

## O

Opinion Mining [198](#)

## P

package janeaustenr [16](#)

packages

about [16](#)

installing [16](#)

installing, CLI used [18](#)  
installing, RStudio GUI used [17](#).  
pie chart  
about [49](#).  
creating [49](#).  
place analysis, Twitter data analysis  
conducting [370](#)  
plotOutput() function [106](#)

## Q

Quote Tweet [344](#).

## R

R Console [10](#)  
retweeting [344](#).  
R language  
about [1](#)  
brief introduction [2](#)  
conditions and loops, programming

RMarkdown  
about [140](#)  
first RMarkdown, writing  
output, generating from [151](#)  
RMarkdown document  
body, creating [142](#)  
creating [139](#).  
R code, embedding [144](#).  
YAML header, setting up [141](#)

R Prompt [10](#)

R software

application, creating [69](#)

installing

invoking [10](#)

obtaining [4](#)

packages [16](#)

setting up [3](#)

RStudio

about [11](#)

installing

interface [14](#)

invoking [15](#)

obtaining [12](#)

setting up [11](#)

rule-based systems, for sentiment analysis [199](#).

**s**

SankhyaSaabdh application

complete code [132](#)

server side, coding [125](#)

UI [114](#)

URL [115](#)

scatter plot

about [50](#)

creating [50](#)

sentiment analysis

about [198](#)

approaches [199](#).

sentiment analysis, conducting  
about [207](#)  
contribution of each word, determining in sentiment score  
sentiment, determining across text [210](#)  
word frequency, generating [209](#)  
words, extracting from text [208](#)  
words with positive and negative sentiment, visualizing  
sentiments, across text  
counts of positive and negative sentiments, separating [213](#)  
net sentiments score for each chunk, calculating [213](#)  
net sentiments score, plotting [214](#)  
sentiment of each line, determining [211](#)  
sentiment of each word, determining [210](#)  
server.R file  
creating, in Shiny application [110](#)  
server-side code, Chidiya  
about [417](#)  
Chidiya, making responsive  
response, initiating to user input

tweets, fetching  
ShabdhMegh application  
complete code [180](#)  
data table output, creating  
developing [151](#)  
download button, programming [177](#)  
file, inputting  
modes, switching between TEXT file and PDF file  
output, creating when input file provided [166](#)  
output, resetting when input file provided [177](#)  
PDF files, inputting [158](#)  
progress bar, displaying [176](#)

radio buttons, creating [156](#)  
screen [152](#)  
server-side code, for inputting PDF files [159](#).  
server-side code, for inputting text file [158](#)  
tabs, creating [154](#)  
TEXT files, inputting [157](#).  
UI [153](#)  
word cloud, displaying [174](#)  
word frequency, computing [168](#)  
world cloud [153](#)  
Shiny application  
about [98](#)  
body of web page [104](#).  
client-side program [103](#)  
creating, RStudio used  
input, accepting [106](#)  
output, displaying [107](#).

publishing  
running [109](#).  
server.R, creating [110](#)  
server-side program [108](#)  
ShabdhMegh, developing [151](#)  
structure [102](#)  
title, displaying [104](#)  
ui.R file, creating [110](#)  
user interface, coding [116](#)  
user interface, designing [114](#)  
Shiny application UI, coding  
images, displaying [117](#).  
layouts, creating [123](#)  
number only input, accepting

text, displaying [118](#)  
text output, displaying [121](#)  
Shiny App Test  
creating [99](#)  
short messaging service (SMS) [485](#)  
sliderInput() function [105](#)  
source analysis, Twitter data analysis  
conducting [375](#)  
statistical approach, of sentiment analysis [199](#)  
statistical operations, on vector [20](#)  
str() function [27](#)

## T

term counting [199](#)  
Term Document Matrix [91](#)  
tidytext library

about [200](#)  
sentiments data set [201](#)  
Trigram analysis  
about [254](#)  
process  
trim() function [72](#)  
tweets  
about [344](#)  
displaying using DT  
displaying with their links [363](#)  
emotion analysis, conducting  
Twitter  
about [324](#)

data, fetching from 345  
favorites 345  
features 344  
graph of growth of Twitter 324  
Hash Tag symbol (#) 345  
logo 325  
Protected Tweets 345  
Quote Tweet 344  
retweeting 344  
tweet 344  
URL 324  
Twitter Developer Account  
2-factor authentication, for logging 327  
about 326  
app approval 336  
app, creating

creating  
home page 327  
home page, for user account 328  
home page, with logged in account 329  
login page 327  
URL, for creating 326

## U

ui.R file  
creating, in Shiny application 110  
user analysis, Twitter data analysis  
conducting  
user interface, Shiny application

coding [116](#)  
designing [114](#).  
UI, for SankhyaSaabdh  
User Interface (UI) [98](#)

## v

vector  
about [18](#)  
assigning, to variables [19](#).  
length of vectors, checking [19](#).  
statistical operations, conducting [20](#)  
vector of characters, creating [18](#)  
vector of integers, creating [19](#).  
vector of strings, creating [19](#).  
vector types, checking [19](#).

## w

### WhatsApp

about [487](#).  
data, downloading from [487](#).  
data, downloading from Android phones [492](#)  
data, downloading from Apple iPhones  
data structure [494](#).  
WhatsApp chat data  
reading, R used [494](#).  
WhatsApp chat data visualization  
creating [495](#).  
Emoji analysis

heat map, for visualizing messages per hour [509](#)  
messages per author, visualizing  
messages per hour, visualizing  
messages per weekday, visualizing [500](#)  
number of messages per day, checking  
radar chart, for visualizing messages per weekday  
word analysis  
word analysis, emotion analysis  
diversity of word frequencies, checking [229](#)  
emotions expressed by words  
most frequently used words in text [231](#)  
statistics [228](#)  
Word Cloud application  
creating [87](#)  
creating, from structured data [89](#)  
creating, from unstructured data  
World Wide Web (WWW) [261](#)

## Y

YAML header

about [141](#)  
setting up [141](#)

## Z

ZEUSg  
about [262](#)  
Basic Information tab [264](#)  
Bi-Gram tab [266](#)

complete code [273](#)  
design [262](#)  
landing page [263](#)  
outputs [264](#)  
Sentiment Analysis tab [265](#)  
Statistics tab [264](#)  
Tri-Gram tab [266](#)  
Word Analysis tab [265](#)  
Word Contribution to Sentiments tab [266](#)  
ZEUSg programming  
about [267](#)  
credit, providing for using library [267](#)  
input file, reading  
libraries, using for development [268](#)  
tabs, creating within tab [269](#)