BIG DATA ANALYTICS USING R

B.A / B.Com (Hons) THIRD YEAR

SEMESTER – V

Lesson Writers

Dr. U. Surya Kameswari
M.Sc CS, M.Tech IT., Ph.D
Assistant Professor
Department of Computer Science and Engineering University College of sciences
Acharya Nagarjuna University

Dr. B. Reddaiah, M.E., Ph.D.
Associate Professor
Department of Computer Science and Technology
Yogi Vemana University
Kadapa-516005

Mr. G V Suresh, M.Tech., (Ph.D)
Associate professor
Department of Computer Science and Engineering Lakireddy Balireddy College of Engineering (Autonomous)
Mylavaram

Mrs. A. Sarvani, M.Tech., (Ph.D)
Associate professor
Department of Information Technology
Lakireddy Balireddy College of Engineering (Autonomous)
Mylavaram

Editor

Dr. K. Lavanya B.E., M.Tech, Ph.D
Assistant professor
Department of Computer Science and Engineering
University College of sciences
Acharya Nagarjuna University
Email: dr.lavanyakampa@gmail.com

Director

Prof.V.VENKATESWARLU
M.A.(Soc), M.S.W., M.Phil., Ph.D.
CENTRE FOR DISTANCE EDUCATION
ACHARYA NAGARJUNA UNIVERSITY
NAGARJUNA NAGAR-522510
website: anucde.info
e-mail: anucdedirector@gmail.com

B.Com (CA): Big Data Analytics Using R

First Edition: 2024

No. of Copies

Published by

Prof.V.Venkateswarlu

Director

Centre for Distance Education

Acharya Nagarjuna University

Nagarjuna Nagar-522510

Printed at

FOREWORD

Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining 'A' grade from the NAAC in the year 2016, Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 443 affiliated colleges spread over the two districts of Guntur and Prakasam.

The University has also started the Centre for Distance Education in 2003-04 with the aim of taking higher education to the door step of all the sectors of the society. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even to housewives desirous of pursuing higher studies. Acharya Nagarjuna University has started offering B.A., and B.Com courses at the Degree level and M.A., M.Com., M.Sc., M.B.A., and L.L.M., courses at the PG level from the academic year 2003-2004 onwards.

To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers involved respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.

It is my aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn be part of country's progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Academic Coordinators, Editors and Lesson-writers of the Centre who have helped in these endeavours.

Prof. K.Gangadhar Rao
Vice-Chancellor
Acharya Nagarjuna University

A.P. State Council of Higher Education
Semester-wise Revised Syllabus under CBCS, 2019-20
Subject: **Computer Applications for Arts/Commerce**
Four year B.A. /B.Com. (Hons) Semester –V (from 2022-23)
Course Code: Max Marks: 100

Course-**6A: BIGDATA ANALYTICS USING R**
(Skill Enhancement Course (Elective), 4 credits)

I. **Learning Outcomes**:

Upon successful completion of the course, a student will be able to:

1. Understand data and classification of digital data.
2. Understand Big Data Analytics.
3. Load data in to R.
4. Organize data in the form of R objects and manipulate them as needed.
5. Perform analytics using R programming.

II. Syllabus: (Total hours: 75 including Theory, Practical, Training, Unit tests etc.)

Unit – 1: Introduction to Big data (12 h)

Data, classification Of Digital Data--structured, unstructured, semi-structured data, characteristics of data, evaluation of big data, definition and challenges of big data , what is big data and why to use big data ?, business intelligence Vs big data.

Unit – 2: Big data Analytics (10 h)

What is and isn't big data analytics? Why hype around big data analytics? Classification of analytics, top challenges facing big data, importance of big data analytics, technologies needed to meet challenges of big data.

Unit – 3: Introduction to R and getting started with R **(**13h)

What is R? Why R? , advantages of R over other programming languages, Data types in R-logical, numeric, integer, character, double, complex, raw, coercion, ls() command, expressions, variables and functions, control structures, Array, Matrix, Vectors, R packages.

Unit – 4: Exploring data in R (13h)

Data frames-data frame access, ordering data frames, R functions for data frames dim(), nrow(), ncol(), str(), summary(), names(), head(), tail(), edit() .Load data frames—reading from .CSV files, sub setting data frames, reading from tab separated value files, reading from tables.

Unit – 5: Data Visualization using R (12h)

Reading and getting data into R (External Data): XML files, Web Data, JSON files, Databases, Excel files.

Working with R Charts and Graphs: Histograms, Bar Charts, Line Graphs, Scatterplots, Pie Charts

BOOKS

1. Seema Acharya , Subhashini Chellappan --- Big Data And Analytics second edition, Wiley
2. Seema Acharya--Data Analytics using R, McGraw Hill education (India) Private Limited.
3. Big Data Analytics, Introduction to Hadoop, Spark, and Machine-Learning, Raj kamal,
   Preeti Saxena, McGraw Hill, 2018.
4. Big Data, Big Analytics: Emerging Business intelligence and Analytic trends for Today's
   Business, Michael Minelli, Michelle Chambers, and Ambiga Dhiraj, John Wiley & Sons,
   2013

Reference Books:

1. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team

RECOMMENDED CO-CURRICULAR ACTIVITIES:

(Co-curricular activities shall not promote copying from textbook or from others work and shall encourage self/independent and group learning)

A. Measurable

1. Assignments (in writing and doing forms on the aspects of syllabus content and outside the syllabus content. Shall be individual and challenging)
2. Student seminars (on topics of the syllabus and related aspects (individual activity))
3. Quiz (on topics where the content can be compiled by smaller aspects and data (Individuals or groups a steams))
4. Study projects (by very small groups of students on selected local real-time problems pertaining to syllabus or related areas. The individual participation and contribution of students shall be ensured (team activity

B. General

1. Group Discussion
2. Try to solve MCQ's available online.
3. Others

RECOMMENDED CONTINUOUS ASSESSMENT METHODS:

Some of the following suggested assessment methodologies could be adopted;

1. The oral and written examinations (Scheduled and surprise tests),
2. Closed-book and open-book tests,
3. Problem-solving exercises,
4. Practical assignments and laboratory reports.
5. Observation of practical skills,
6. Individual and group project reports like "Creating Text Editor in C".
7. Efficient delivery using seminar presentations,
8. Viva voce interviews.
9. Computerized adaptive testing, literature surveys and evaluations,
10. Peers and self-assessment, outputs form individual and collaborative work

Course-6A: **Big Data Analytics Using R---- Lab (Practical) Syllabus (15 Hrs.)**

(Since, the proposed SECs are connected to Computer Programming/Software Tools and Skill enhancement, the students need to get exposure on the syllabus content by practicing on the computer even though there is no formal assignment of credits and laboratory hours for practical sessions. So, as part of the Co-curricular activities and continuous assessment, students should be engaged in practicing on computer for at least 15 hours per semester.)

1. Create a vector in R and perform operations on it.
2. Create integer, complex, logical, character data type objects in R and print their values and their class using print and class functions.
3. Write code in R to to demonstrate sum(), min(), max() and seq() functions.
4. Write code in R to manipulate text in R using grep(), toupper(), tolower() and substr() functions.
5. Create data frame in R and perform operations on it.
6. Import data into R from text and excel files using read.table () and read.csv () functions.
7. Write code in R to find out whether number is prime or not.
8. Print numbers from 1 to 100 using while loop and for loop in R.
9. Write a program to import data from csv file and print the data on the console.
10. Write a program to demonstrate histogram in R.

Note: The list of experiments need not be restricted to the above list. *Detailed list of Programming/software tool based exercises can be prepared by the concerned Faculty members.*

BIGDATA ANAYLSTICS USING R
CONTENTS

LESSON- 1

OVERVIEW OF DATA

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the importance of data
- Describe the classification of data.
- Explain why data is vital to success in today's world
- Discuss the characteristics of data.

STRUCTURE OF THE LESSION:

1.1 WHAT IS DATA

Data, in simple terms, refers to pieces of information. Imagine data as small building blocks of information that computers use to do all sorts of things, like showing pictures on a screen, solving math problems, or sending messages to friends. These blocks can be numbers, like scores in a game or temperatures outside, or they can be words, like the text in a book or an email you write. Data can also be things we see and hear, like photos or music. Basically, data is anything that computers can understand and work with to help us do tasks, learn new things, and have fun.

Data in a Computer is a stream of bits (0s and 1s) that are saved in computer memory. These bits of information can take the shape of text documents, images, videos, etc. The CPU (Central Processing Unit) performs this data processing and stores it in the computer's memory. As a result, data in the computer can be generated, processed, and saved

Data in computers refers to encoded information that computers process and manipulate to perform various tasks. It can take various forms, including text, numbers, multimedia files, and more. Essentially, data is the raw material that computers use to generate meaningful outputs through algorithms and computations.

In simply way, data is a collection of raw facts.

Data → information

Information → insight

1.2 TYPES OF DATA

1.2.1 Numerical data:

One common form of data in computers is numeric data, which consists of numerical values used for calculations, measurements, and statistical analysis. For example, in a spreadsheet application, numeric data might include sales figures, inventory quantities, or financial metrics. These numbers can be manipulated using mathematical operations to derive insights or make informed decisions.

Numeric data consists of numerical values used for quantitative analysis and calculations. Examples include:

- Sales figures
- Temperature readings
- Stock prices
- Sensor measurements (e.g., voltage, pressure, temperature)

1.2.2 Text data:

Another type of data prevalent in computers is text data, which comprises characters, words, sentences, and paragraphs. Text data is ubiquitous in documents, emails, web pages, and other textual content. For instance, in a word processing software, text data could include the contents of a report, a letter, or a blog post. Computers use algorithms for text processing tasks such as spell checking, grammar correction, and text summarization.

Text data consists of characters, words, sentences, and paragraphs used to convey information. Examples include:

- Documents (e.g., reports, articles, essays)
- Emails
- Social media posts
- Website content
- Chat transcripts

1.2.3 Multimedia data:

Furthermore, computers handle multimedia data, which encompasses images, audio, video, and other multimedia content. Multimedia data is prevalent in entertainment, communication, education, and various other domains. For instance, in a photo editing application, multimedia data might include digital images that users edit, enhance, or manipulate using various tools and filters.

Multimedia data includes images, audio, video, and other forms of media. Examples include:

- Digital photographs
- Music files (e.g., MP3, WAV)
- Video clips (e.g., MP4, AVI)
- Animations

- Virtual reality simulations

1.2.4 Real-time data

Moreover, computers process real-time data, which refers to data generated or updated continuously, often in response to external events or sensor inputs. Real-time data is critical in applications such as financial trading, monitoring systems, and IoT (Internet of Things) devices. For example, in a weather forecasting system, real-time data from weather sensors and satellites is continuously analyzed to predict weather patterns and issue warnings. Overall, data in computers encompasses a wide range of formats and types, driving various applications and innovations in the digital age.

Real-time data is generated or updated continuously and requires immediate processing or analysis. Examples include:

- Stock market data
- Weather sensor readings
- GPS location data
- Social media streams
- IoT (Internet of Things) sensor data

1.3 REPRESENTATION OF DATA IN COMPUTERS

We have all seen computers do seemingly miraculous things with all kinds of sounds, pictures, graphics, numbers, and text. It seems that we can build a replica of parts of our world inside the computer. You might think that this amazing machine is also amazingly complicated - it really is not. In fact, all of the wonderful multi-media that we see on modern computers is all constructed from simple ON/OFF switches - millions of them - but really nothing much more complicated than a switch. The trick is to take all of the real-world sound, picture, number etc data that we want in the computer and convert it into the kind of data that can be represented in switches, as shown in Figure 1.

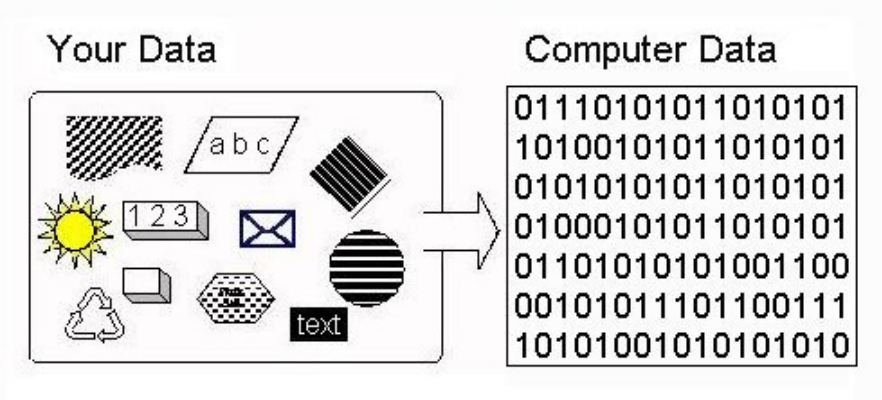


Figure 1.1 data conversion

Computers Are Electronic Machines. The computer uses electricity, not *mechanical* parts, for its data processing and storage. Electricity is plentiful, moves very fast through wires, and electrical parts fail less much less frequently than mechanical parts. The computer does have some mechanical parts, like its disk drive (which are often the sources for computer failures), but the internal data processing and storage is electronic, which is fast and reliable (as long as the computer is plugged in).

Electricity can flow through switches: if the switch is closed, the electricity flows; if the switch is open, the electricity does not flow. To process real-world data in the computer, we need a way to represent the data in switches. Computers do this representation using a *binary coding system*.

Binary and Switches. Binary is a mathematical number system: a way of counting. We have all learned to count using ten digits: 0-9. One probable reason is that we have ten fingers to represent numbers. The computer has switches to represent data and switches have only two states: ON and OFF. Binary has two digits to do the counting: 0 and 1 - a natural fit to the two states of a switch (0 = OFF, 1 = ON).

Bits and Bytes One binary digit (0 or 1) is referred to as a *bit*, which is short for ***bi**nary digi**t***. Thus, one bit can be implemented by one switch

A single byte can represent many different kinds of data. What data it actually represents depends on how the computer ues the byte.

1.4 CLASSIFICATION OF DIGITAL DATA

1.4.1 Structured Data

Structured data is generally tabular data that is represented by columns and rows in a database. Databases that hold tables in this form are called ***relational databases***. The mathematical term "*relation"* specifies a formed set of data held as a table. In structured data, all row in a table has the same set of columns. SQL (Structured Query Language) programming language used for structured data.

Structured data refers to information that is organized in a predefined and easily understandable format, typically within a database or spreadsheet. This format involves organizing data into categories, tables, or fields, with each piece of information labeled and arranged systematically. Structured data can include various types of data, such as numerical values, dates, text, and identifiers. Examples of structured data include databases of customer information, inventory lists, financial records, and schedules.

Structured data is data that has a standardized format for efficient access by software and humans alike. It is typically tabular with rows and columns that clearly define data attributes. Computers can effectively process structured data for insights due to its quantitative nature. For example, a structured customer data table containing columns—name, address, and phone number—can provide insights like the total number of customers and the locality with the maximum number of customers. In contrast, unstructured data, like a list of social media posts, is more challenging to analyze.

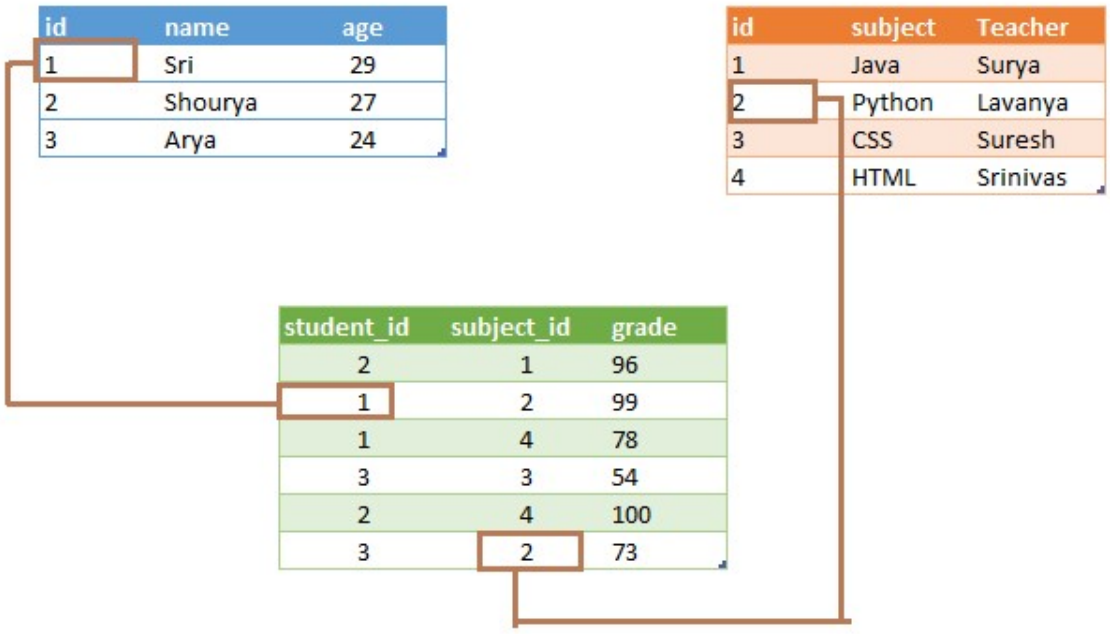


Figure 1.2 Structured Data

1.4.1.1 Benefits of structured data

There are several benefits of using structured data.

Ease of use

Anyone can quickly comprehend and access structured data. Operations such as updating and amending structured data are straightforward. Storage is efficient, as fixed-length storage units can be allocated for data values.

Scalability

Structured data scales algorithmically. You can add storage and processing power as your data volume increases. Modern systems that process structured data can scale to several thousand TB of data.

Analytics

Machine learning algorithms can analyze structured data and identify common patterns for business intelligence. You can use structured query language (SQL) to generate reports as well as modify and maintain data. Structured data is also useful for big data analytics.

1.4.1.2 Challenges of structured data

While there are several advantages of using structured data for business, there are also some challenges.

Limited usage

The predefined structure is a benefit but can also be a challenge. Structured data can only be utilized for its intended purpose. For example, booking data can give you information about booking system finances and booking popularity. But it can't reveal which marketing campaigns were more effective in bringing in more bookings without further modification. You'll have to add marketing campaign relational data to your bookings if you want the additional insights.

Inflexibility

It can be costly and resource-intensive to change the schema of structured data as circumstances change and new relationships or requirements emerge.

1.4.2 Semi-Structured Data

Semi-structured data is information that doesn't consist of structured data (relational database) but still has some structure to it. Semi-structured data consists of documents held in *JavaScript Object Notation* (JSON) format. It also includes *key-value* stores and *graph* databases.

Semi-structured data is also referred to as self-describing structure. This is the data which does not conform to a data model but has some structure. However, it is not in a form which can be used easily by a computer program. About 10% data of an organization is in this format; for example, HTML, XML, JSON, email data etc

Semi-structured data refers to information that does not conform to the strict structure of traditional structured data (e.g., databases with fixed schemas) but still has some organizational properties that make it more structured than unstructured data. In semi-structured data, elements within the dataset may have irregularities or variations in their format, yet they are organized in a way that retains some level of hierarchy or tagging. This type of data often lacks a rigid schema but may contain metadata or tags that provide some level of organization or context.

One common example of semi-structured data is XML (eXtensible Markup Language), which allows users to define their own tags to structure data hierarchically. While XML documents may not adhere to a predefined schema like traditional databases, they still possess a degree of structure due to the nested nature of elements and the use of tags to delineate different types of information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Arya</to>
  <from>Shourya</from>
  <heading>Reminder</heading>
  <body>Don't forget to attend the meeting</body>
</note>
```

Figure 1.3 Sample XML File

Another example is JSON (JavaScript Object Notation), a lightweight data interchange format commonly used in web development. JSON data is organized into key-value pairs and nested objects, providing a flexible way to represent structured data in a format that is easy for humans to read and machines to parse. While JSON does not enforce a rigid schema, it typically follows a consistent structure within a given dataset.

```
{
    "glossary": {
        "title": "example glossary",
                "GlossDiv": {
            "title": "S",
                        "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                                        "SortAs": "SGML",
                                        "GlossTerm": "Standard Generalized Markup Language",
                                        "Acronym": "SGML",
                                        "Abbrev": "ISO 8879:1986",
                                        "GlossDef": {
                        "para": "A meta-markup language, used to create markup languages such as DocBook.",
                                                "GlossSeeAlso": ["GML", "XML"]
                    },
                                        "GlossSee": "markup"
                }
            }
        }
    }
}
```

Figure 1.4 Sample JSON File

Semi-structured data presents both challenges and opportunities for data management and analysis. On one hand, its flexible nature allows for capturing and storing diverse types of information without the constraints of a fixed schema. On the other hand, processing and analyzing semi-structured data may require specialized tools and techniques to handle variations in format and structure. Nonetheless, semi-structured data is increasingly prevalent in modern data environments, particularly in contexts such as web data, IoT (Internet of Things) devices, and social media feeds, where flexibility and adaptability are paramount.

1.4.3 Unstructured Data

Unstructured data refers to information that does not have a predefined data model or organization, making it more challenging to analyze and process using traditional methods. Unlike structured data, which is organized into rows and columns or follows a specific schema, unstructured data lacks a consistent format and may include text documents, emails, images, videos, social media posts, audio recordings, and more.

One characteristic of unstructured data is its variability in content and structure, as it can contain free-form text, multimedia files, or combinations thereof. This variability makes it difficult to apply traditional database techniques for storage and retrieval. Additionally, unstructured data often contains rich and valuable information, such as customer feedback, market trends, or insights from multimedia content, but extracting meaning from it requires advanced techniques like natural language processing, image recognition, or sentiment analysis.

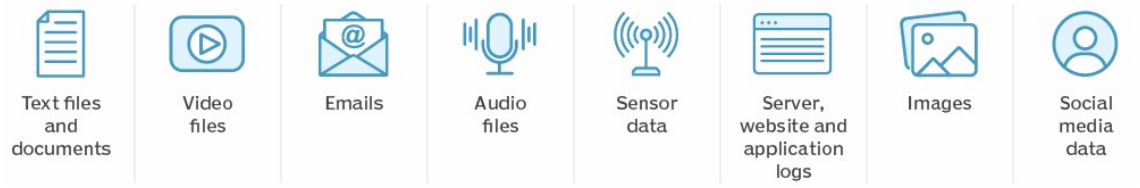


Figure 1.5unstructured data types

Examples of unstructured data include legal documents, audio, chats, video, images, text on a web page, and much more. Discover some of the most common unstructured data examples below:

Examples of unstructured data are:

- Business Documents
- Emails
- Social Media
- Customer Feedback
- Webpages
- Open Ended Survey Responses
- Images, Audio, and Video

1.5 CHARACTERISTICS OF A DATA

Data has three key characteristics:

1.5.1 Composition:

Structure: Data composition refers to the arrangement and organization of elements within the dataset. Structured data has a predefined format with clear relationships between its components, such as tables in a database or fields in a spreadsheet. In contrast, unstructured data lacks a fixed structure and may contain text, images, videos, or other multimedia content with varying formats and arrangements.

Granularity: Granularity describes the level of detail or specificity present in the data. Fine-grained data contains highly detailed information at a low level of abstraction, while coarse-grained data provides broader summaries or aggregates at a higher level of abstraction.

Dimensionality: Dimensionality refers to the number of attributes or features that characterize each data point. High-dimensional data has a large number of variables or dimensions, which can pose challenges for visualization, analysis, and interpretation.

1.5.2 Condition:

Quality: Data condition relates to its quality and integrity. High-quality data is accurate, complete, consistent, and free from errors or biases that could impact its reliability or usefulness. Data quality measures include validity, accuracy, completeness, consistency, and timeliness.

Freshness: The freshness of data indicates how recently it was collected or updated. Fresh data reflects the most current information available and is crucial for making timely decisions and insights.

Currency: Currency refers to the relevance and applicability of the data within a specific timeframe or context. Currency is essential for ensuring that data remains relevant and actionable for decision-making processes.

1.5.3 Context:

Relevance: Data relevance pertains to its significance and applicability to the problem or task at hand. Relevant data provides valuable insights and answers specific questions, aligning closely with the user's objectives and needs.

Interpretation: Contextual information, such as metadata, annotations, or documentation, helps users interpret and understand the meaning of the data. Contextual cues provide additional context, background, or insights that aid in the interpretation and analysis of the data.**Usage Context**: The usage context refers to the environment or scenario in which the data is utilized. Understanding the usage context helps tailor data analysis, visualization, and interpretation to meet the specific requirements and expectations of users, ensuring the relevance and effectiveness of the insights derived from the data.

These characteristics collectively influence how data is collected, stored, processed, analyzed, and interpreted to derive insights and support decision-making processes. By considering data composition, condition, and context, organizations can better understand and leverage the value of their data assets for achieving their objectives and gaining a competitive advantage.

Small data (data as it existed prior to the big data revolution) is about certainty. It is about known data sources; it is about no major changes to the composition or context of data.

Most often we have answers to queries like why this data was generated, where and when it was generated, exactly how we would like to use it, what questions will this data be able to answer, and so on. Big data is about complexity. Complexity in terms of multiple and unknown datasets, in terms of exploding volume, in terms of speed at which the data is being generated and the speed at which it needs to be processed and in terms of the variety of data (internal or external, behavioural or social) that is being generated

1.6 SUMMARY

This chapter delves into the fundamental concepts of data and its various types, including structured, unstructured, and semi-structured data. Data is defined as raw facts or information that can be processed to obtain meaningful insights. Understanding the different types of data is essential for effective data management and analysis in contemporary society. Understanding the distinctions between structured, unstructured, and semi-structured data is crucial for modern data management and analysis practices. Each type of data presents unique challenges and opportunities, and leveraging appropriate technologies and methodologies is essential for deriving actionable insights from diverse data sources.

1.7 TECHNICAL TERMS

Big data, structured data, unstructured data, semi-structured data

1.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Define data and discuss its significance in contemporary society. How does understanding the various types of data

2. What are the distinguishing characteristics of structured, unstructured, and semi-structured data? Provide examples of each type.
3. Discuss the significance of semi-structured data in the context of modern data ecosystems. How do formats like JSON, XML, and RDF facilitate interoperability and integration across diverse data sources and systems?

Short Notes:

1. What is data?
2. What are the main types of data?
3. What is structured data?
4. What is unstructured data?
5. What is semi-structured data?

1.9 SUGGESTED READINGS

1. Seema Acharya , Subhashini Chellappan --- Big Data And Analytics secondedition, Wiley
2. 2. Seema Acharya--Data Analytics using R, McGraw Hill education (India) Private Limited.
3. 3. Big Data Analytics, Introduction to Hadoop, Spark, and Machine-Learning, Rajkamal, Preeti Saxena, McGraw Hill, 2018.
4. 4. Big Data, Big Analytics: Emerging Business intelligence and Analytic trends forToday's Business, Michael Minelli, Michelle Chambers, and Ambiga Dhiraj, John Wiley &Sons, 2013

AUTHOR: **Dr. U. Surya Kameswari**

LESSON- 2

INTRODUCTION TO BIG DATA

OBJECTIVES:

After going through this lesson, you will be able to

- Understand what is Big Data
- Understand Applications of Big Data
- Known challenges in Big Data
- Gain knowledge about characteristics of big data.

STRUCTURE OF THE LESSION:

2.1 EVALUATOIN OF BIG DATA

The quantity of data created by humans is quickly increasing every year as a result of the introduction of new technology, gadgets, and communication channels such as social networking sites.Big data is a group of enormous datasets that can't be handled with typical computer methods. It is no longer a single technique or tool; rather, it has evolved into a comprehensive subject including a variety of tools, techniques, and frameworks. Quantities, letters, or symbols on which a computer performs operations and which can be store

The history of big data starts many years before the present buzz around Big Data. Seventy years ago the first attempt to quantify the growth rate of data in the terms of volume of data was encountered. That has popularly been known as *"information explosion"*.
The following are the some major milestones in the evolution of "big data".
1944:

Fremont Rider, based upon his observation, speculated that Yale Library in 2040 will have "approximately 200,000,000 volumes, which will occupy over 6,000 miles of shelves…a cataloguing staff of over six thousand persons."

From 1944 to 1980, many articles and presentations were presented that observed the 'information explosion' and the arising needs for storage capacity.
1980:

In 1980, the sociologist Charles Tilly uses the term big data in one sentence "none of the big questions has actually yielded to the bludgeoning of the big-data people." in his article "The old-new social history and the new old social history".

But the term used in this sentence is not in the context of the present meaning of Big Data today.

Now, moving fast to 1997-1998 where we see the actual use of big data in its present context.
1997:

In 1977, Michael Cox and David Ellsworth published the article "Application-controlled demand paging for out-of-core visualization" in the Proceedings of the IEEE 8th conference on Visualization.

The article uses the big data term in the sentence "Visualization provides an interesting challenge for computer systems: data sets are generally quite large, taxing the capacities of main memory, local disk, and even remote disk. We call this the problem of big data. When data sets do not fit in main memory (in core), or when they do not fit even on local disk, the most common solution is to acquire more resources."

It was the first article in the ACM digital library that uses the term big data with its modern context.

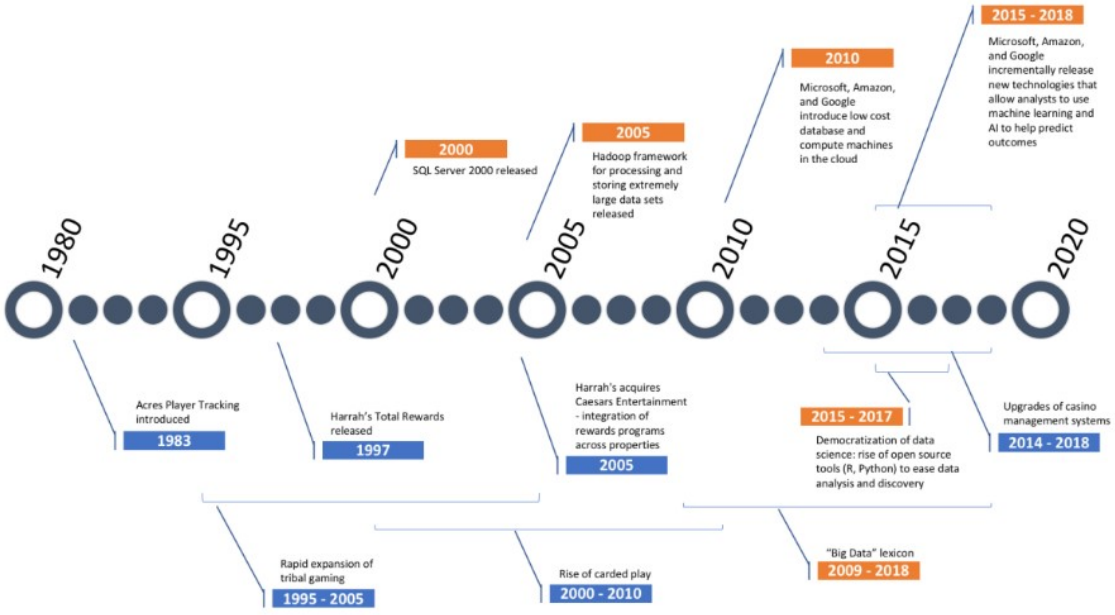


Figure 2.1 Evolution of big data *1998:*

In 1998, John Mashey, who was Chief Scientist at SGI presented a paper titled "Big Data… and the Next Wave of Infrastress." at a USENIX meeting. John Mashey used this term in his various speeches and that's why he got the credit for coining the term Big Data.
2000:

In 2000, Francis Diebold presented a paper titled "' Big Data' Dynamic Factor Models for Macroeconomic Measurement and Forecasting" to the Eighth World Congress of the Econometric Society.

In the paper, he stated that "Recently, much good science, whether physical, biological, or social, has been forced to confront—and has often benefited from—the "Big Data" phenomenon.

Big Data refers to the explosion in the quantity (and sometimes, quality) of available and potentially relevant data, largely the result of recent and unprecedented advancements in data recording and storage technology."
2001:

In 2001, Doug Laney, who was an analyst with the Meta Group (Gartner), presented a research paper titled "3D Data Management: Controlling Data Volume, Velocity, and Variety." The 3V's have become the most accepted dimensions for defining big data.
2005:

In 2005, Tim O'Reilly published his groundbreaking article "What is Web 2.0?". In this article, Tim O'Reilly states that the "data is the next Intel inside".

O'Reilly Media explicitly used the term 'Big Data' to refer to the large sets of data which is almost impossible to handle and process using the traditional business intelligence tools.
This is for sure the current widely understood form of big data definition.

In 2005 Yahoo used Hadoop to process petabytes of data which is now made open-source by Apache Software Foundation. Many companies are now using Hadoop to crunch Big Data.
So we can say that 2005 is the year that the big data revolution has truly begun and the rest they say is history.

2.2 DEFINITION: BIG DATA

The definition of big data can vary depending on the context and the author's perspective.
However, a commonly cited definition of big data was proposed by Doug Laney, a research vice president at Gartner, in 2001. This definition is often referred to as the "3Vs" model, which highlights three key characteristics of big data: volume, velocity, and variety.
Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.

2.3 CHALLENGES OF BIG DATA

The major implementation barriers are going to be the challenges that Big Data presents. These require quick attention and need to be treated since, if they are not handled, the technology may fail, which can also lead to certain undesirable results. If they are not handled, then the technology may fail. The issues associated with big data include the storage and analysis of data that is both incredibly huge and rapidly expanding.

1. The necessity of attaining synchronization across a variety of data sources
There is a significant obstacle to overcome in order to incorporate data sets into an analytical platform since data sets are consistently growing in size and variety. In the event that this is neglected, it will result in gaps, which will in turn lead to incorrect signals and insights.

2. A severe lack of knowledgeable professionals who are able to analyses large amounts of data
In order to make the vast amount of data that is being produced every minute meaningful, it is essential to do data analysis. The industry has seen a significant increase in the demand for big data scientists and big data analysts as a result of the exponential growth of data. Due to the fact that the job of a data scientist encompasses a wide range of disciplines, it is essential for businesses to select a data scientist who possesses a diverse set of talents. There is a shortage of personnel that are knowledgeable with Big Data analysis, which is another significant obstacle that firms must overcome. There is a significant lack of data scientists in relation to the enormous amount of data that is being produced.

3. Obtaining Meaningful Insights through the Application of Big Data Analytics
When it comes to corporate organizations, it is of the utmost importance to acquire valuable insights through the utilization of Big Data analytics. Additionally, it is additionally essential that only the right department has access to this information. When it comes to Big Data analytics, one of the most significant challenges that businesses confront is finding an efficient way to close this considerable gap.

4. Bringing Massive Amounts of Data into the Big Data Technology Platform
It should not come as a surprise that the amount of data is increasing with each passing day. A simple interpretation of this would be that corporate organizations are required to manage a substantial volume of data on a daily basis. The sheer volume and variety of data that is currently available can be overwhelming for any data engineer. For this reason, it is believed to be of the utmost importance to make data accessibility simple and convenient for entrepreneurs and managers of brands.

5. The unpredictability of the landscape of data management
Every single day, new businesses and technologies are being launched as a direct result of the proliferation of big data. Companies that are involved in Big Data analytics, on the other hand, are confronted with a significant obstacle, which is determining which technology will be most suitable for them without introducing any new problems or potentially dangerous situations.

6. The Storage of Data and Its Quality
There has been a remarkable increase in the number of business organizations. Companies and other large commercial organizations are experiencing enormous expansion, which has led to an increase in the amount of data that is being produced. Every single person is facing a significant obstacle in the form of the storage of this enormous volume of data. Frequently utilized data storage methods, such as data lakes and warehouses, are utilized for the purpose of collecting and storing vast volumes of unstructured and structured data in the format that it was originally stored in. When a data lake or warehouse attempts to combine unstructured and inconsistent data from a variety of sources, it runs into mistakes. This is the fundamental problem that develops. There are a number of factors that can lead to problems with data quality, including missing data, inconsistent data, logic conflicts, and duplicate data.

7. Protection of Personal Information and Data Security
Once commercial organizations learn how to make use of big data, it opens up a wide variety of chances and possibilities for them. Nevertheless, it also contains the potential concerns that are connected with big data, particularly with regard to the privacy of the data and the security of the network. There are many different sources of data that are utilized by the Big Data technologies that are utilized for analysis and storage. In the long run, this results in a high danger of the data being exposed, which makes it susceptible to being compromised. Concerns regarding privacy and security are intensified as a result of the proliferation of large amounts of data.

2.4 WHAT IS BIG DATA AND WHY TO USE BIG DATA

The relevance of big data does not depend on the amount of data that a company possesses; rather, it is determined by how the organization makes use of the data that it has acquired. Each and every business makes use of data in its own unique manner; the more effectively a firm makes use of its data, the greater the potential for growth it possesses. The organization is able to collect data from any source and do analysis on it in order to discover solutions that will enable:

1. Cost Savings: When huge amounts of data need to be stored, certain Big Data tools, such as Hadoop and Cloud-Based Analytics, can bring cost advantages to businesses. These tools also assist in determining more effective ways of conducting business, which provides additional cost savings.

2. Time Reductions: The fast speed of tools such as Hadoop and in-memory analytics makes it simple to recognize new sources of data, which enables organizations to analyses data rapidly and make decisions based on the learning in a short amount of time.

3. Gain an insight of the current market conditions: We can benefit from a better grasp of the current market conditions by doing an analysis of big data. An example of this would be a corporation performing an analysis of the purchase patterns of its clients in order to determine which products are the most popular and then producing products in accordance with this trend. By doing so, it is able to gain an advantage over its rivals.

4. Management of one's internet reputation: instruments that work with big data can perform sentiment analysis. That being the case, you are able to obtain input regarding who is saying what about your organization. In the event that you are interested in monitoring and enhancing the internet presence of your company, then big data tools might be of assistance to you in all of this.

5. Increasing the Retention and Acquisition of Customers: Through the Use of Big Data Analytics The client is the most crucial asset that any business depends on. There is not a single company that can safely assert that they have achieved success without first needing to build up a strong consumer base. The high level of competition that a company faces is something that it cannot afford to ignore, even if it already has a customer base. In the event that a company is sluggish to learn what it is that its clients are seeking for, it is quite simple for that company to start delivering products of inferior quality. When all is said and done, the loss of customers will be the end outcome, which will have a negative impact on the

overall success of the firm. With the use of big data, businesses are able to discover a variety of patterns and trends that are relevant to their customers. In order to instill loyalty in customers, it is essential to observe their behavior.

6. The application of big data analytics to solve problems faced by advertisers and provide marketing insights:

All corporate activities can be altered with the assistance of big data analytics. The ability to meet the expectations of customers, the modification of the product line offered by the company, and, of course, the guarantee that marketing campaigns are effective are all included in this.

7. **Big Data Analytics as a Driver of Innovations and Product:** Development One further significant advantage of big data is its capacity to assist businesses in the process of innovating and redeveloping their products.

2.5 BUSINESS INTELLIGENCE VS BIG DATA

2.5.1 Business Intelligence:

The term "Business Intelligence" (BI) refers to the utilization of various technologies, applications, and techniques for the purpose of gathering, integrating, analyzing, and presenting business data. In order to facilitate more effective decision-making around commerce, Commerce Insights was developed. Trade Insights frameworks are, in essence, Decision Support Systems (DSS) that are driven by analysis of data. The term "business intelligence" is sometimes used interchangeably with "briefing books," "reports and inquiry instruments," and "official data frameworks." The majority of the time, business intelligence frameworks make use of information that has been compiled into a data warehouse or a data store, and they may occasionally work from operational data. These frameworks provide authentic, up-to-date, and forward-looking perspectives of business operations.

Business Intelligence has the following advantages:

Business intelligence (BI) is primarily concerned with the provision of insights that are derived from historical data. This enables companies to comprehend patterns and trends in their operations.

By providing a comprehensive perspective of the activities of the organization, business intelligence (BI) enables managers to comprehend performance across a variety of departments and roles for the organization.

BI can assist in the identification of opportunities for the reduction of costs and the optimization of processes, which can ultimately lead to enhanced efficiency and profitability. Some of the drawbacks of using business intelligence:

Due to the fact that business intelligence is centered on past data, it may not provide an accurate picture of the current or future conditions.

A large investment in data gathering and processing, in addition to specialized software and

hardware, is required for business intelligence (BI), which can be a resource-intensive endeavor.

The level of depth or granularity that is required to address particular business difficulties might not be provided by business intelligence.

2.5.2 Differences between Business Intelligence and Big Data

There are distinctions between Big Data and Business Intelligence, despite the fact that both of these technologies are utilized to analyses data in order to assist businesses in the process of decision-making. Both the manner in which they operate and the kinds of data that they analyses are distinct from one another.

The gathering, examination, and presentation of data for the purpose of facilitating strategic decision-making inside an organization are the three main components that comprise business intelligence technology. Extraction of data from a variety of sources, transformation of that data into information that is relevant, and presentation of that information in the form of reports, dashboards, and visualizations are all components of this process. Through the use of business intelligence, executives, managers, and analysts are given the ability to monitor key performance indicators, recognize patterns, and acquire insights that can be put into action in order to ensure the growth and efficiency of their organizations.

On the other hand, the term "Big Data" refers to enormous quantities of data that are particularly complicated and varied, and which cannot be simply managed by the conventional methods of data processing. It incorporates both structured and unstructured data from a variety of sources, including electronic transactions, social media platforms, sensors, and other sources. In order to draw important insights and find patterns, correlations, and trends that were not previously recognized, Big Data utilizes advanced analytics and processing methods. These approaches include data mining, predictive analytics, and machine learning. It gives businesses the ability to investigate opportunities that have not yet been exploited, learn about the preferences of their customers, improve their decision-making processes, and optimize their operations. Business Intelligence and Visualization courses will assist students in transforming data into opportunities through the use of BI and Visualization capabilities, as well as in preparing themselves for employment.

Data Type

Business Intelligence (BI) typically deals with structured data, which is organized and categorized into preset formats such as databases, spreadsheets, and data warehouses. This type of data involves the organization and categorization of information. This type of structured data is often generated internally within an organization and adheres to a schema that has been predefined. Using the tools and methods that are traditionally used for data processing, it is simple to query, aggregate, and analyses the data.

Big Data comprises a wide variety of data formats, including organized, semi-structured, and unstructured variations of the same piece of information. Information that does not correspond to a fixed schema or a predefined format is included in Big Data, in addition to the structured data that is utilized in business intelligence. Included in this are posts on social

media platforms, reviews from customers, emails, photos, videos, sensor data, and other types of content. The applications of Big Data technology make it possible to store, process, and analyses a wide variety of data kinds.

Volume of Data

Business intelligence (BI) often works with moderate to large data sets that can be managed by employing the tools and procedures that are traditionally used for data processing. When it comes to business intelligence (BI), the size of the data volumes often falls within the range of what can be kept in a relational database or processed using traditional data warehouses. Extracting insights from relevant subsets of data in order to help decision-making is the primary focus of business intelligence (BI).

Big Data is a term that describes datasets that are so enormous that they exceed the capabilities of conventional database management systems to process them. The volume of Big Data, which is typically measured in terabytes, petabytes, or even bigger scales, is one of its defining characteristics. The burden is distributed across numerous servers or clusters of machines in order to facilitate the storing, processing, and analysis of these enormous datasets. This is made possible by Big Data technology.

Data Sources:

The majority of business intelligence (BI) relies on the data sources that are already present within an organization. Transactional databases, enterprise resource planning (ERP) systems, data warehouses, customer relationship management (CRM) systems, spreadsheets, and other structured data repositories are some of the sources that fall under this category. In most cases, the data that is utilized in business intelligence is produced by the organization's very own systems and applications.

Big Data comprises a wider variety of data sources than traditional data does. This system combines data from external sources in addition to data from internal sources. These external sources include social networking platforms (such as Twitter and Facebook), weblogs, machine sensors, geolocation data, public datasets, and other sources. Big Data places an emphasis on collecting and analyzing data from a wide variety of sources in order to achieve a more comprehensive perspective and more profound understanding.

An Approach to Analysis

The primary objective of business intelligence (BI) is to collect and examine structured data through the utilization of methods such as reporting, querying, and data visualization. Extracting insights from structured information is accomplished through the utilization of well-established techniques such as online analytical processing (OLAP) and data mining. Business intelligence (BI) tools and platforms offer users interactive dashboards, reports, and visualizations to assist them in comprehending the data and making decisions that are driven by the data.

The examination of big data goes beyond the usual business intelligence methods. Data mining, machine learning, natural language processing, and predictive analytics are some of the advanced analytics technologies that are utilized and utilized by this system. Hadoop, Spark, and Nosily databases are examples of Big Data technologies that make it possible to process enormous datasets that are not simple to understand. In the context of business intelligence (BI), Big Data analytics seeks to identify patterns, correlations, and trends that may not be immediately obvious using more conventional methods. Exploratory analysis,

anomaly identification, and predictive modelling are frequently utilized in this process in order to get more profound understandings from extensive and unstructured data.

Purpose:
Within an organization, business intelligence is largely utilized to provide assistance with the process of making operational decisions. The ability to provide insights into corporate processes, performance metrics, and operational efficiency is the primary focus of this component. Through the use of business intelligence (BI), stakeholders such as executives, managers, and analysts are able to monitor and analyses the performance of the business, locate areas that could be improved, and make decisions that are informed by both historical and currently available data.

The goal of Big Data analysis is to unearth useful insights and to obtain a deeper understanding of complicated processes. This objective drives the study. The purpose of this endeavor is to unearth previously concealed patterns, trends, and anomalies that have the potential to result in innovations, new possibilities, and strategic decision-making. Big data is centered on the exploration of data in order to derive insights that can be put into action and to drive corporate change. The ability to identify new markets, optimize operations, enhance customer experiences, and create products and services are all some of the benefits that it may provide to organizations.

Time Sensitivity: Business Intelligence (BI) includes both real-time and historical analysis. Users are able to monitor and analyses data in real-time or near-real-time with the help of real-time business intelligence, which helps them gain immediate insights into ongoing operations. Real-time dashboards, for instance, are able to display live data on key performance indicators. This enables stakeholders to monitor metrics and take rapid actions based on information that is current at the moment. In business intelligence, historical analysis is the process of looking at data from the past in order to recognize patterns, trends, and historical performance over a certain period of time.

Big Data processing frequently comprises data processing and analysis that is performed in real time or very close to real time. When organizations have the capability to handle and analyses massive amounts of data in a short amount of time, they are able to make choices in a timely manner and take prompt action based on emerging patterns or occurrences. Streaming data from sensors or social media, for instance, can be processed and analyzed in real time in order to identify abnormalities, keep track of trends, or initiate automatic responses.

A User's Role
A wide variety of users, such as executives, managers, analysts, and decision-makers across a variety of business departments, are intended to make advantage of business intelligence's capabilities. Users are able to access relevant data, generate reports, construct interactive dashboards, and run ad hoc searches with the help of business intelligence (BI) tools and platforms, which provide user-friendly features and intuitive interfaces. Business intelligence makes it possible for people with diverse degrees of technical expertise to investigate data, increase their level of insight, and work together on decision-making processes. In most cases, the analysis of Big Data calls for specialized knowledge and abilities, and it

frequently involves the participation of data scientists, analysts, and researchers. Experts in advanced analytics techniques, statistics, programming, and data manipulation, these people are extremely knowledgeable in these areas. They work with datasets that are both huge and complicated, and they extract important insights by applying machine learning algorithms, data mining approaches, and statistical models. The analysis of big data frequently entails the participation of multidisciplinary teams that work together to identify insights and formulate suggestions based on the data.

2.6 CHARACTERISTICS OF BIG DATA

"Big Data" is data whose scale, diversity, and complexityrequire new architecture, techniques, algorithms, andanalytics to manage it and extract value and hiddenknowledge from it.

The characteristics of big data, encapsulated by the three Vs - Volume, Velocity, and Variety - along with additional dimensions like Veracity, Variability, and Value, underscore its profound impact on modern analytics and decision-making.

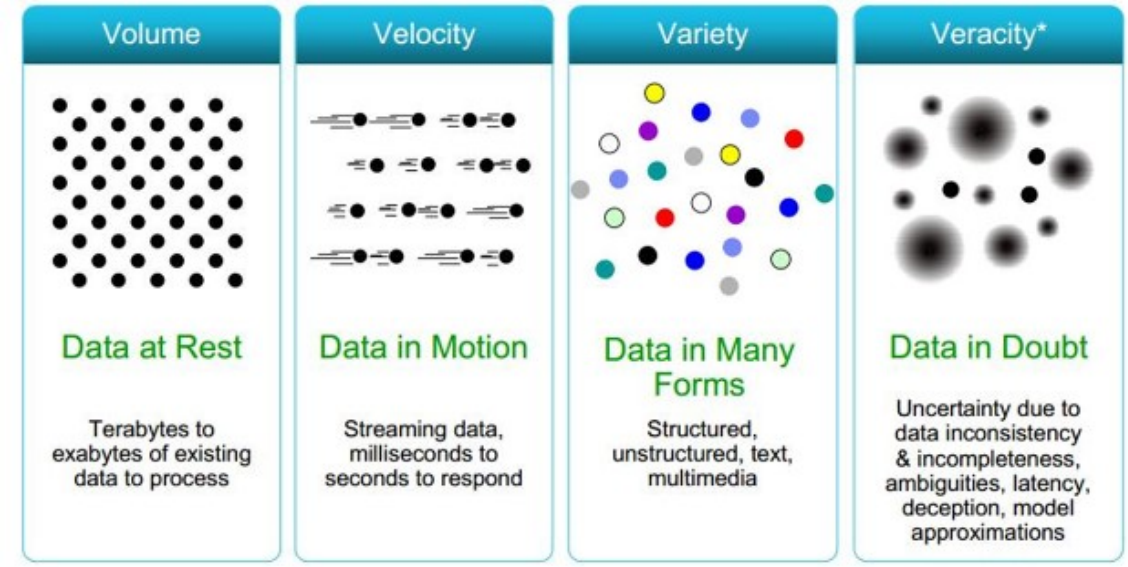


Figure 2.2 Characteristics of Big Data

2.6.1. Volume

The volume of within the context of the current situation, the quantity of data that businesses possess is significant. It will be necessary for you to analyze larger amounts of structured and unstructured data in order to perform major data analytics. These datasets, such as those found on Facebook and Instagram, as well as the data found on a variety of web and mobile applications, can have an endless amount of value. The volume of data is expected to significantly increase in the next years, as shown by the trends in the industry, and there is a great deal of potential for extensive data analysis and the discovery of patterns through this process.

2.6.2.Velocity

The speed at which data is processed is indicated by the term "velocity." When it comes to the real-time evaluation and performance of any big data activity, getting a higher data processing rate is really important. In the future, users will have access to a greater quantity

of data; nevertheless, the processing speed will be of equal significance for businesses who wish to reap the benefits of big data analytics.

2.6.3. Variety

Variety is a term that relates to the various classifications of big data. Because of the influence it has on productivity, it is one of the most significant difficulties that the big data sector faces.
The increasing utilization of big data has resulted in the emergence of new data groups. It is necessary to perform additional pre-processing on various data categories, such as text, audio, and video, in order to support metadata and determine increased value.

2.6.4 Value

Value refers to the benefits that your organization derives from the data that has been processed and examined. It communicates how the data corresponds to the goals that have been established for your organization and whether or not it helps your firm improve itself. In the realm of big data, it is among the most essential basic qualities.

2.6.5 Veracity

Veracity is a term that refers to the accuracy of your data. It is crucial because of the potential for low veracity to have a detrimental impact on the accuracy of the outcomes of your big data analytics.

2.6.6. Validity

The validity of the data is a measure of how useful and relevant information is for a firm to use in order to achieve the goals that have been envisioned and the purpose that has been established.

2.6.7. Volatility

Changes are constantly occurring in big data. The knowledge that you have gathered from a specific source at this moment can be different in a short period of time. Inconsistency in the data is indicated by this scenario, which also has an effect on the rate at which you accommodate and adapt to the data.

2.6.8. Visualization

Visualization, also known as data visualization, is the process of displaying the analytics and insights that have been generated by your big data through the use of visual drawings like as charts and graphs. As professionals in big data share their analytics and insights with non-technical addressees, it has become increasingly significant.

2.7 SUMMARY

The chapter begins by defining big data, emphasizing its three key characteristics: volume, velocity, and variety. It explores the significance of big data in contemporary society, highlighting its role in enabling organizations to derive valuable insights and make data-driven decisions. The challenges of big data, including managing its sheer volume, velocity, and variety, are discussed in detail, along with considerations of data quality, privacy, and

security. Additionally, the chapter compares big data with traditional business intelligence, illustrating how big data extends beyond structured data analysis to encompass diverse data types and advanced analytics techniques. By examining the evaluation, definition, challenges, and distinctions between big data and business intelligence, the chapter provides a comprehensive overview of the complexities and opportunities associated with harnessing big data for organizational success.

2.8 TECHNICAL TERMS

Big Data, Business Intelligence

2.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Identify and analyze the major challenges associated with big data
2. Compare and contrast traditional business intelligence (BI) with big data analytics
3. Provide examples of real-world applications where big data has been instrumental in driving transformation and creating value.

Short Questions:

1. What are the key characteristics used to define big data according to the traditional "3Vs" model?
2. What are some of the challenges organizations face when dealing with big data?
3. Explain the concept of big data and its significance in modern business contexts.
4. How does big data differ from traditional business intelligence approaches?

2.10 SUGGESTED READINGS

1. Seema Acharya , SubhashiniChellappan --- Big Data And Analytics secondedition, Wiley
2. 2. Seema Acharya--Data Analytics using R, McGraw Hill education (India) Private Limited.
3. 3. Big Data Analytics, Introduction to Hadoop, Spark, and Machine-Learning, Rajkamal, PreetiSaxena, McGraw Hill, 2018.
4. 4. Big Data, Big Analytics: Emerging Business intelligence and Analytic trends forToday's Business, Michael Minelli, Michelle Chambers, and AmbigaDhiraj, John Wiley &Sons, 2013

AUTHOR: **Dr. U. Surya Kameswari**

LESSON- 3

BIG DATA ANALYTICS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand what is and what is not Big Data Analytics
- Understand classification of Big Data Analytics
- Known the challenges in Big Data Analytics
- Explain the importance of big data analytics.

STRUCTURE OF THE LESSION:

3.1 WHAT IS AND ISN'T BIG DATA ANALYTICS?

However, the RDBMS and the typical data warehouse are not going to be replaced by big data analytics. Data warehouses and relational database management systems are coexisting with big data analytics. It is not possible to characterize something as "big data" if it just consists of a massive volume of data. Extremely high volumes are a characteristic, but volume alone is not sufficient to justify big data. That is not to say that only large corporations make use of it; any company can make use of it.

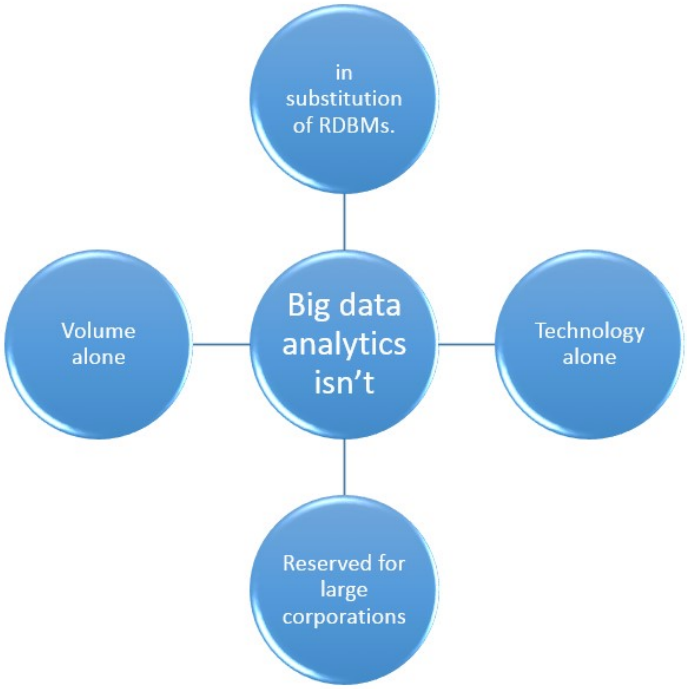


Figure 3.1 What isn't big data analytics

3.1.1 Rather than RDBMs.

Relational Database Management Systems (RDBMS) and big data don't directly replace each other. Instead, big data is a complementary tool that meets different data management needs. RDBMS work well in traditional database settings for storing structured data and processing transactions. They give you a solid and organised way to handle data with clear schemas, and their ACID (Atomicity, Consistency, Isolation, and Durability) features make sure that the data is correct. However, RDBMS might not be able to handle the huge amounts of data that modern applications create, as well as the variety and speed of data that is created in today's digital world.

Big data systems, on the other hand, let you handle large amounts of structured, semi-structured, and unstructured data in a way that is both scalable and flexible. Distributed computing systems are used by these platforms, like Hadoop and Spark, to process and analyse data across groups of standard hardware. The three Vs of big data are volume, velocity, and variety, and big data tools are great at handling all three. They let businesses keep, process, and look at very large datasets for things like real-time data processing, predictive analytics, and machine learning. There are some things that big data platforms can do that standard RDBMS can't, but they don't completely replace them. Instead, they add an extra layer to handle use cases and data types that RDBMS might not be able to handle well on its own.

A method that combines both RDBMS and big data technologies to use the best features of each is often used by businesses. RDBMS are still needed for transactional processing, which makes sure that data is correct and consistent in important business processes. Meanwhile, big data platforms store and analyse huge amounts of data, which lets businesses get useful

information from many different sources of data. By putting these tools together, businesses can make a complete data management plan that covers all of their data needs, from structured transactional data to big data analytics on a large scale.

3.1.2 Volume alone.

One big difference between big data and RDBMS is how they handle large amounts of data. RDBMS can handle organized data that fits into schemas that have already been set up. The size of this data is usually between a gigabyte and a terabyte. They work well with traditional transactional systems that handle small amounts of data and put a lot of stress on keeping data correct and consistent. But RDBMS might not be able to handle data as well after a certain point, especially as the amount of data grows into the petabyte or exabyte range.

Big data systems, on the other hand, are designed to handle huge amounts of data, from terabytes to petabytes and even more. You can use distributed storage and processing systems on these platforms to make them work with more cheap hardware in groups. It is easy for big data platforms to store, process, and analyze huge datasets because they break down data processing jobs into smaller, parallelizable units. They can handle a lot of data, which makes them perfect for web-scale analytics, real-time data processing, and large-scale machine learning. Traditional RDBMS would have a hard time keeping up with all that data.

3.1.3 Technology alone.

There are big technological changes between Relational Database Management Systems (RDBMS) and big data platforms. These differences show how they handle data differently. RDBMS use a centralized architecture to keep data in structured tables with predefined schemas. This is usually done on a single server or a small group of servers. Structured Query Language (SQL) is used to change and query data in these systems, which are designed to handle transactions quickly and keep data safe. Big data platforms, on the other hand, use distributed computer architectures that let hardware clusters of the same type be used to add more resources. They are made to handle very large amounts of data, like petabytes or exabytes, and can process data across many nodes at the same time. Big data platforms can handle structured, semi-structured, and unstructured data, as well as other data formats and structures. This means they can handle the wide range of data sources that are common in current applications.

Also, big data technologies often use different structures and tools for processing data than RDBMS. RDBMS use SQL a lot to explore and change data, but big data platforms use distributed computing frameworks like Flink, Hadoop, and Spark. These tools make it possible to do machine learning, batch processing, and stream processing in real time. Big data platforms also often include NoSQL databases, like HBase and Cassandra, which make it easy to store and view semi-structured and unstructured data. Big data platforms can change with the times thanks to their wide range of technologies. They can be used for many things, from basic analytics to more complex methods like deep learning and natural language processing.

3.1.4 Reserved for large corporations.

Relational Database Management Systems (RDBMS) and big data are both used by corporations for different reasons in their data management plans. Companies use RDBMS to handle organized data in transactional systems like OLTP, CRM, and ERP. Because these systems make sure that data is correct, consistent, and dependable, they can be used for important business tasks where structured data needs to be saved, accessed, and changed quickly. RDBMS are often the most important part of business applications that need to process data and handle transactions in real time.

In contrast, big data technologies are being used more and more by businesses to deal with the problems that come up with handling huge amounts of data from sources like social media, sensor data, log files, and multimedia material. Big data platforms, such as Hadoop, Spark, and NoSQL databases, offer scalable storage and processing. This lets businesses collect, store, and analyze big datasets to gain insights and make decisions. Big data is used by businesses for jobs like fraud detection, customer segmentation, predictive analytics, and sentiment analysis. Traditional relational databases may not be able to handle the amount, type, and speed of data that is involved. Companies can get useful information from many different types of data thanks to big data tools. This helps them make better business decisions and gain market share.

3.2 WHY HYPE AROUND BIG DATA ANALYTICS?

The hype around big data analytics stems from several factors that collectively contribute to its significance and appeal in various industries:

As digital tools have become more common, the amount of data that businesses produce has grown by a huge amount. Big data analytics lets you handle and learn from these huge datasets, which lets you access useful data that you couldn't get to before. Big data analytics can work with different kinds of data, such as organized, semi-structured, and unstructured data. This includes things like text, pictures, videos, posts on social media, sensing data, and more. By looking at this wide range of data, businesses can learn a lot about their processes, customers, and markets. Big data analytics lets businesses look at data in real time or very close to real time. This skill is very important in environments that are always changing and need quick insights to adjust to new issues, market trends, or changes in customer behavior.

By finding patterns and trends in large datasets, big data analytics makes predictive modeling and predicting easier. This helps businesses guess what will happen in the future, make smart choices, and successfully lower their risks. Companies can make their goods, services, and marketing more relevant to each customer by looking at data about them from different sources. Big data analytics helps businesses understand what customers want, how they act, and how they feel, which lets them make products that fit each customer's needs and makes them happier.

Cost savings can come from using big data analytics to improve resource distribution, streamline processes, and cut down on waste. Big data analytics also gives companies a competitive edge by letting them make decisions based on data, come up with new ideas faster, and stay ahead of market trends. Big data analytics has become more useful as technologies like machine learning, artificial intelligence, and natural language processing

have improved. These technologies make it possible to analyze data in more complex ways, automate chores, and get more useful information from data.

Overall, the hype surrounding big data analytics shows how it has the ability to change the way businesses grow, come up with new ideas, and gain a competitive edge in today's data-driven world. As companies keep putting money into big data analytics, the hype should turn into real results and broad use across all fields.

Big Data isn't just a bunch of talk; it's a chance for people who know what they're doing. Some people are using analytics, rules engines, and machine learning on Big Data to make data research and search tools, even though this is still in its early stages. Big Data will change the game in more ways than one if it is used in the right way, with the right strategy, with knowledge of the business, and with the right technologies to back it up.

3.3 CLASSIFICATION OF ANALYTICS

Big data analytics encompasses various types of analysis techniques aimed at extracting insights, patterns, and valuable information from large and diverse datasets. Some of the key types of big data analytics include:

Collecting, processing, and interpreting data to find insights and help with making choices is an important part of the field of data analytics. Data analytics is the study of looking at large amounts of raw data to find patterns, draw conclusions, and get useful information. To do this, different methods and tools are used to process and turn data into useful information that can be used to make decisions.

Data analytics includes a lot of different ways to look at data and find useful information that can make different parts of a business better. Businesses can find patterns and metrics that they might not have seen otherwise by carefully looking at data. This lets them improve general efficiency and make processes run more smoothly.

In manufacturing, for example, companies keep track of machine runtime, breaks, and work queues to better plan their workloads and make sure machines work at their best.

Data analytics is used in many areas besides just optimizing output. Companies that make games use analytics to come up with reward systems that keep players interested. Companies that make content use analytics to improve the placement and appearance of their content, which in turn keeps users interested.

There are four types of Data analytics

1. Predictive Analytics (What Happened in the past)
2. Descriptive Analytics (Why did it happened in the Past)
3. Prescriptive Analytics (What will happen in the Future)
4. Diagnostic Analytics (How can we make it happen)

3.3.1 Predictive Analytics

With predictive analytics, the data is turned into useful information that can be used. Predictive analytics uses data to figure out what will probably happen or how likely it is that something will happen. A lot of different statistical methods, like modeling, machine learning, data mining, and game theory, are used in predictive analytics to look at past and present events and guess what will happen in the future.

Descriptive analytics involves summarizing historical data to gain insights into past events and understand what has happened. It focuses on identifying patterns, trends, and correlations within the data. Descriptive analytics provides a foundation for further analysis and decision-making.

Techniques that are used for predictive analytics are Linear Regression, Time Series Analysis and Forecasting, Data Mining

Basic Cornerstones of Predictive Analytics are Predictive modelling, Decision Analysis and optimization, Transaction profiling

3.3.2 Descriptive Analytics

Descriptive analytics looks at data and events that happened in the past to figure out how to handle events that will happen in the future. Looking at past performance and understanding performance by mining previous data to find out why things worked or didn't work in the past. This kind of research is used in almost all management reports, like those for sales, marketing, operations, and finances.

The descriptive model counts the connections between pieces of data in a way that is often used to put customers or leads into groups. A predictive model tries to guess how one customer will act, but descriptive analytics looks at all the possible connections between a customer and a product.

Common examples of Descriptive analytics are company reports that provide historic reviews like Data Queries, Reports, Descriptive Statistics, Data dashboard

3.3.3 Prescriptive Analytics

Prescriptive analytics takes predictive analytics a step further by providing recommendations or actions to optimize outcomes. It uses optimization and simulation techniques to suggest the best course of action based on predicted future scenarios. Prescriptive analytics helps organizations make data-driven decisions and take actions that lead to desired outcomes.

Prescriptive analytics automatically combine large amounts of data, math, business rules, and machine learning to make a prediction. It then offers a decision that can be made based on the prediction.

Prescriptive analytics does more than just guess what will happen in the future. It also suggests actions that will help the predictions come true and shows the person making the decision what each choice will mean. This type of analytics not only guesses what will happen and when, but also tries to figure out why it will happen. Prescriptive analytics can also offer different ways to make decisions about how to take advantage of a future

opportunity or reduce a future risk, and it can show what each choice would mean.

For example, Prescriptive Analytics can benefit healthcare strategic planning by using analytics to leverage operational and usage data combined with data of external factors such as economic data, population demography, etc.

3.3.4 Diagnostic Analytics

For this research, we usually use historical data over other data to find the answer to any question or figure out how to fix any issue. We look at the problem's past data to see if there are any patterns or dependencies.

For instance, businesses choose this type of analysis because it helps them understand a problem better and keep thorough records on what they have available. If they don't, collecting data for each problem individually would take a lot of time.

Diagnostic analytics goes beyond descriptive analytics to answer the question of why certain events occurred. It aims to uncover the root causes of specific outcomes or trends observed in the data. By analyzing historical data in detail, diagnostic analytics helps organizations understand the factors influencing their performance or business operation

Common techniques used for Diagnostic Analytics are Data discovery, Data mining, Correlations

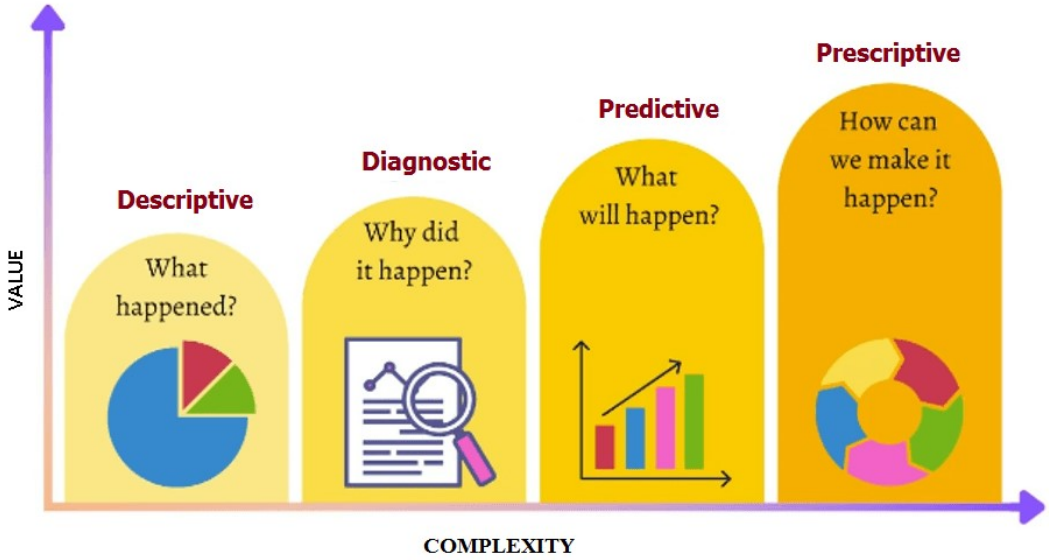


Figure 3.2 Types of Data Analytics

3.4 TOP CHALLENGES FACING BIG DATA

Storage:

The hardest part is storing all the data that is created every day, especially when it comes in different forms, in old systems. Databases that are used for organized data can't hold unstructured data.

Processing:

Processing "big data" means reading, changing, extracting, and organizing data so that it can be used. There are still problems with putting and receiving information in unified forms.

Security: Organizations care a lot about security. Computer thieves could steal or damage information that isn't secured. So, people who work in data security have to find a balance between letting people access data and following strict security rules.

Finding and Fixing Problems with Data Quality

A lot of you are probably having problems because your data isn't very good, but there are ways to fix them.

There are three ways to fix problems with data, which are:

- The original database has the right details.
- Any mistakes in the data must be fixed by fixing the original data source.
- To figure out who someone is, you must use very accurate methods.

Scaling Big Data Systems

Moving to the cloud, splitting read-only and write-active databases, database sharing, and memory caching are all good ways to scale. They are all great on their own, but when you use them together, you'll get to the next level.

Evaluation and Selecting Big Data Technologies

Big data technologies cost a lot of money for businesses, and the market for these tools is growing very quickly. But in the last few years, the IT business has realized how useful big data and analytics can be.

The following technologies are on the rise:

- Hadoop Ecosystem
- Apache Spark
- NoSQL Databases
- R Software
- Predictive Analytics
- Prescriptive Analytic

Figure 3.3 Big Data Challenges

Environments for Big Data

A large data set is more active than a data warehouse because it is always getting new data from different sources. The people who are in charge of big data will quickly forget where and what each set of data came from.

Real-Time insights

By analyzing data as it is being collected by a system, "real-time analytics" describe the process. Real-time analytics tools use logic and math to quickly give views on data, which helps people make decisions more quickly and with more accurate information.

Validation of Data

It is important to make sure that data is correct, complete, and organized before using it in a business process. What comes out of a data validation process can be used for more research, business intelligence, or even to teach a machine learning model what to do.

Healthcare Challenges

Big data about health can come from a lot of different places, such as electronic health records (EHRs), genomic sequencing, medical studies, wearable tech, and medical imaging.

Problems that make it hard to use big data effectively in healthcare

- The price of implementation
- Compiling and polishing data
- Security
- Disconnect in communication

3.5 IMPORTANCE OF BIG DATA ANALYTICS

It's impossible to overstate how important big data analytics is to businesses today; it's a key part of driving growth, making better decisions, and getting a competitive edge. First, big data analytics helps businesses find useful information in the huge amounts of data they create and store. Companies can get useful business information from this data by looking for patterns, trends, and correlations. These insights help businesses learn more about their customers, markets, and operations, which helps them make better goods and services that meet the needs of their customers.

Big data analytics makes it easier to make decisions based on data, which means that companies can choose based on facts and analysis instead of gut feelings or guesswork. Organizations can predict future trends, risks, and opportunities by using predictive analytics. This lets them deal with problems before they happen and take advantage of new possibilities. This proactive method makes companies more flexible and resilient, which helps them stay ahead of the curve in markets that change quickly.

Big data analytics helps businesses make better use of their resources and processes. Companies can find inefficiencies, streamline workflows, and boost output by looking at data on processes, supply chains, and resource use. This optimization not only cuts costs but also improves operational efficiency. This lets businesses give their customers better goods and services more quickly and easily.

Big data analytics is a key part of making the customer experience better and getting them more involved. Companies can learn a lot about their customers' likes, dislikes, and feelings by looking at data from a lot of different places, like social media, website exchanges, and transaction histories. With this information, businesses can tailor their goods, services, and advertising to each customer, giving them more relevant and interesting experiences that make them loyal and happy.

Lastly, big data analytics helps businesses come up with new ideas and stay competitive in the digital world we live in now. Companies can find new possibilities, make new products and services, and set themselves apart from competitors by using advanced analytics techniques like machine learning, artificial intelligence, and deep learning. Big data analytics also lets businesses try new things and make changes quickly, which lets them respond to

changing market conditions and take advantage of new possibilities as they appear. Overall, big data analytics is necessary for businesses that want to do well in a world that is becoming more and more data-driven. It helps them learn important things, make smart choices, improve customer experiences, and spur innovation and growth.

1.6 SUMMARY

The chapter delves into the realm of big data analytics, delineating its essence and clarifying what constitutes, as well as what doesn't, fall under its purview. It explores the hype surrounding big data analytics, attributing it to the transformative potential it holds in unlocking valuable insights from vast and varied datasets, driving innovation, and fostering competitive advantage. Classification of analytics is elucidated, delineating descriptive, diagnostic, predictive, prescriptive, text, spatial, and streaming analytics, each catering to distinct data analysis objectives and methodologies. The chapter also scrutinizes the top challenges facing big data analytics, ranging from data quality and security issues to the complexity of integrating disparate data sources. Ultimately, it underscores the paramount importance of big data analytics in today's digital landscape, elucidating its pivotal role in enhancing decision-making, optimizing operations, fostering innovation, and augmenting customer experiences, thereby enabling organizations to thrive in an increasingly data-driven world.

1.7 TECHNICAL TERMS

Big Data Analytics, Predictive Analytics, Prescriptive Analytics, Descriptive Analytics, Diagnostic Analytics

1.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Discuss the factors contributing to the hype surrounding big data analytics
2. Define and classify different types of analytics
3. Identify and discuss the key challenges and obstacles associated with big data analytics,
4. Evaluate the role of big data analytics in addressing complex business challenges.

Short Questions:

1. What factors contribute to the hype surrounding big data analytics?
2. Describe the characteristics and applications of descriptive analytics.
3. How does predictive analytics differ from prescriptive analytics?
4. Why is big data analytics important for businesses and organizations?
5. What are some of the major challenges organizations face in implementing big data analytics

1.9 SUGGESTED READINGS

1. Seema Acharya , Subhashini Chellappan --- Big Data And Analytics secondedition, Wiley

2. 2. Seema Acharya--Data Analytics using R, McGraw Hill education (India) Private Limited.

3. 3. Big Data Analytics, Introduction to Hadoop, Spark, and Machine-Learning, Rajkamal, Preeti Saxena, McGraw Hill, 2018.

4. 4. Big Data, Big Analytics: Emerging Business intelligence and Analytic trends forToday's Business, Michael Minelli, Michelle Chambers, and Ambiga Dhiraj, John Wiley &Sons, 2013

AUTHOR: **Dr. B. Reddaiah**

LESSON- 4

BIG DATA TECHNOLOGIES

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of big data and its significance in today's data-driven world.
- Recognize the various technologies available for handling big data effectively.
- Explain the role of NoSQL databases in addressing the challenges of big data storage and processing.

STRUCTURE OF THE LESSION:

4.1 No SQL

The acronym NoSQL was first used in 1998 by Carlo Strozzi while naming his lightweight, open-source “relational” database that did not use SQL. The name came up again in 2009 when Eric Evans and Johan Oskarsson used it to describe non-relational databases. Relational databases are often referred to as SQL systems. The term NoSQL can mean either “No SQL systems” or the more commonly accepted translation of “Not only SQL,” to emphasize the fact some systems might support SQL-like query languages.

Big data is a term used to describe the enormous amount of data that organizations produce on a daily basis. Previously, the size and complexity of this data exceeded the capabilities of conventional data processing techniques. Several large data processing tools are accessible, such as Apache Hadoop, Apache Spark, and MongoDB. Each of these technologies possesses distinct advantages and disadvantages, although all of them can be utilized to extract valuable information from extensive data sets. Big data storage technologies refer to a computational and storage framework that is designed to gather and handle vast amounts of data. These technologies also enable the analysis of data in real-time.

Figure 4.1 Big Data Technologies

Big data encompasses a wide range of technologies and tools that are used to store, process, analyze, and visualize large and complex datasets. Some of the key technologies used in big data include:

Hadoop: Hadoop is an open-source framework for distributed storage and processing of large datasets across clusters of commodity hardware. It includes components such as Hadoop Distributed File System (HDFS) for storage and MapReduce for parallel processing.

Apache Spark: Apache Spark is a fast and general-purpose distributed computing system that provides in-memory processing capabilities for big data analytics. It supports a wide range of programming languages and APIs for batch processing, streaming, machine learning, and graph processing.

NoSQL Databases: NoSQL (Not Only SQL) databases are non-relational databases designed for scalable, high-performance storage and retrieval of unstructured and semi-structured data. Examples include MongoDB, Cassandra, HBase, Couchbase, and Redis.

Apache Kafka: Apache Kafka is a distributed streaming platform that provides high-throughput, fault-tolerant messaging for real-time data processing and analytics. It is commonly used for building real-time data pipelines and event-driven architectures.

Apache Flink: Apache Flink is a stream processing framework for distributed, high-throughput, and low-latency data processing. It supports event time processing, exactly-once semantics, and stateful computations for real-time analytics.

Data Warehousing Solutions: Data warehousing solutions such as Amazon Redshift, Google BigQuery, and Snowflake provide scalable, cloud-based platforms for storing and analyzing structured data. They are optimized for high-performance analytics and business intelligence applications.

Machine Learning and AI: Machine learning and artificial intelligence techniques are used to extract insights, patterns, and predictions from big data. Frameworks such as TensorFlow, PyTorch, and scikit-learn are commonly used for building and deploying machine learning models on big data.

Data Visualization Tools: Data visualization tools such as Tableau, Power BI, and D3.js are used to create interactive visualizations and dashboards for exploring and communicating insights from big data.

Distributed Storage Systems: Distributed storage systems such as Amazon S3, Google Cloud Storage, and Apache HDFS provide scalable and durable storage for big data. They are optimized for storing large volumes of data across distributed nodes.

Containerization and Orchestration: Containerization technologies such as Docker and container orchestration platforms such as Kubernetes are used to deploy and manage big data applications in scalable and portable environments.
These technologies form the foundation of modern big data ecosystems, enabling organizations to harness the full potential of their data for decision-making, innovation, and competitive advantage.

4.1.1TYPES OF No SQL DATA BASES

A database is a collection of structured data or information which is stored in a computer system and can be accessed easily. A database is usually managed by a Database Management System (DBMS).
NoSQL is a non-relational database that is used to store the data in the nontabular form. NoSQL stands for Not only SQL. The main types are documents, key-value, wide-column, and graphs.
Types of NoSQL Database:

- Document-based databases
- Key-value stores
- Column-oriented databases
- Graph-based databases

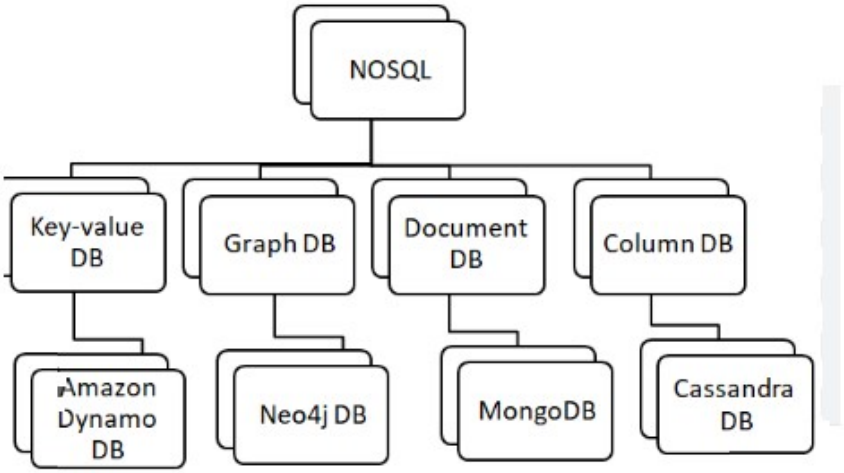


Figure 4.2 Types of No SQL

4.1.1.1 Document-Based Database:

The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON, or XML documents.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications. In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

Collections are the group of documents that store documents that have similar contents. Not all the documents are in any collection as they require a similar schema because document databases have a flexible schema.

Key features of documents database:

- Flexible schema: Documents in the database has a flexible schema. It means the documents in the database need not be the same schema.
- Faster creation and maintenance: the creation of documents is easy and minimal maintenance is required once we create the document.
- No foreign keys: There is no dynamic relationship between two documents so documents can be independent of one another. So, there is no requirement for a foreign key in a document database.
- Open formats: To build a document we use XML, JSON, and others.

4.1.1.2 Key-Value Stores:

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a key-value store. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.

A key-value store is like a relational database with only two columns which is the key and the value.

Key features of the key-value store:

- Simplicity.
- Scalability.
- Speed.

4.1.1.3 Column Oriented Databases:

A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.

Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data.

Key features of columnar oriented database:

- Scalability.
- Compression.
- Very responsive.

4.1.1.4 Graph-Based databases:

Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships.

Key features of graph database:

- In a graph-based database, it is easy to identify the relationship between the data by using the links.
- The Query's output is real-time results.
- The speed depends upon the number of relationships among the database elements.
- Updating data is also easy, as adding a new node or edge to a graph database is a straightforward task that does not require significant schema changes.

4.1.2 ADVANTAGES OF No SQL

Scalability:

NoSQL databases are designed for horizontal scalability, allowing organizations to handle large volumes of data by adding more nodes to the cluster.

They distribute data across multiple nodes in a cluster, enabling parallel processing and efficient scaling without sacrificing performance.

Flexibility:

NoSQL databases offer flexible data models, including key-value, document, columnar, and graph databases, allowing organizations to choose the most suitable model for their data.

They support dynamic schema design, enabling agile development and adaptation to changing data requirements without the need for schema migrations.

Performance:

NoSQL databases are optimized for high-performance data storage and retrieval operations, leveraging distributed architectures and parallel processing techniques.

They provide low-latency query responses and high-throughput data processing, making them well-suited for real-time analytics and high-volume transactional systems.

Availability and Fault Tolerance:

NoSQL databases prioritize availability and fault tolerance, ensuring continuous access to data even in the event of hardware failures or network partitions.
They employ replication, sharding, and automatic failover mechanisms to distribute data and workload across multiple nodes and ensure data redundancy and resilience.

Schema Evolution:

NoSQL databases support dynamic schema evolution, allowing organizations to adapt to changing data requirements and evolving business needs seamlessly.

They enable flexible schema design and schema-less approaches, simplifying data modeling and application development and facilitating agile development and iteration.

Diverse Data Types:
No SQL databases can handle various types of data, including structured, semi-structured, and unstructured data, without the need for predefined schemas or data mappings.

They support diverse data models and data types, enabling organizations to store and process different types of data efficiently within a single database system.

Cost-Effectiveness:
NoSQL databases can be more cost-effective than traditional relational databases, particularly in large-scale deployments.

They utilize commodity hardware and open-source software, reducing infrastructure costs and licensing fees compared to proprietary relational database solutions.

Ease of Development:
NoSQL databases offer simplified data modeling and application development, with support for modern programming languages and development frameworks.

They provide flexible APIs and developer-friendly tools, enabling rapid prototyping, experimentation, and iteration in application development.

These advantages make NoSQL databases a preferred choice for organizations seeking scalable, flexible, and high-performance solutions for managing large and diverse datasets in today's data-intensive environments.

4.1.3 USE OF NoSQL IN INDUSTRY

1. E-commerce: - NoSQL databases are employed for the management of product catalogs, customer profiles, and transactional data in e-commerce systems.
- They provide immediate inventory control, customized suggestions, and streamlined order fulfillment, thereby boosting the overall client shopping experience.

2. Social Media: - NoSQL databases are utilized in social media platforms to store user profiles, postings, comments, and interactions.
No information provided. These platforms provide the processing of large amounts of data, the analysis of data in real-time, and the delivery of customized content. This allows social media firms to expand their operations and interact with millions of users at the same time.

3. Gaming: - NoSQL databases are used in gaming systems for tasks such as managing player profiles, handling game state, and doing in-game analytics.
No information provided. They provide the capacity to easily increase in size without interruption, allow for quick access to data with minimal delay, and enable immediate changes to leaderboards, so improving the gaming experience and allowing interactions between several players.

4. IoT (Internet of Things): - NoSQL databases are used to manage sensor data, telemetry streams, and device logs that are created by IoT devices in different sectors like manufacturing, healthcare, and smart cities.

- They provide assistance for analyzing data in real-time, identifying anomalies, and conducting predictive maintenance. This allows enterprises to efficiently monitor and improve their IoT deployments.

5. Financial Services:- NoSQL databases are utilized in financial services to handle customer transactions, conduct risk analytics, and detect fraudulent activities.

No information provided. They provide the immediate processing of financial data, reporting on compliance, and analysis of trends, so assisting enterprises in reducing risks and improving operational efficiency.

6. Healthcare: - NoSQL databases are utilized in healthcare systems to hold electronic health records (EHRs), medical imaging data, and patient demographics.

- They provide assistance for the secure exchange of data, the ability for different systems to work together, and the analysis of data in real-time to aid in making clinical decisions, monitoring diseases, and managing the health of populations.

7. Telecommunications: - NoSQL databases manage call detail records (CDRs), network logs, and subscriber data in telecommunications networks.

- They facilitate high-volume data processing, network optimization, and consumer segmentation, empowering telecom firms to enhance service quality and customer happiness.

8. AdTech (Advertising Technology): - AdTech platforms utilize NoSQL databases to handle ad impressions, clickstream data, and user profiles.

There is no text provided. Their platform facilitates real-time bidding, precise ad targeting, and campaign optimization, empowering marketers to deliver tailored advertisements to specific audiences and optimize their return on investment.

These examples demonstrate the diverse applications of NoSQL databases in different industries to meet varied data management and analytics requirements. NoSQL databases offer the necessary scalability, flexibility, and performance to manage extensive and varied information in the current data-centric era.

4.1.4 NoSQL VENDORS

MongoDB: MongoDB is a popular document-oriented NoSQL database that provides flexible schema design, scalability, and high performance. It is widely used for a variety of use cases, including content management, real-time analytics, and mobile applications.

Cassandra (Apache Cassandra): Cassandra is a distributed, wide-column store NoSQL database designed for high availability, fault tolerance, and linear scalability. It is well-suited for applications requiring real-time data processing, such as IoT, financial services, and messaging platforms.

Couchbase: Couchbase is a distributed NoSQL database that combines key-value and document database capabilities. It provides low-latency data access, automatic data sharding, and support for distributed caching, making it suitable for high-performance applications, caching layers, and real-time analytics.

Amazon Dynamo DB: Dynamo DB is a fully managed NoSQL database service provided by Amazon Web Services (AWS). It offers seamless scalability, high availability, and low-latency data access, making it ideal for web and mobile applications, gaming, and IoT use cases.

Google Cloud Firestore: Firestore is a flexible, scalable NoSQL database service offered by Google Cloud Platform (GCP). It supports real-time data synchronization, offline support, and automatic scaling, making it suitable for mobile and web applications, collaborative workflows, and IoT deployments.

Azure Cosmos DB: Cosmos DB is a globally distributed, multi-model NoSQL database service provided by Microsoft Azure. It supports multiple data models, including document, key-value, graph, and column-family, and offers automatic scaling, low-latency data access, and multi-region replication, making it suitable for a wide range of applications and industries.

Redis: Redis is an in-memory data store NoSQL database that provides high-performance data caching, session management, and real-time analytics capabilities. It is commonly used for caching layers, message brokers, and real-time data processing in applications such as e-commerce, gaming, and social media.

Neo4j: Neo4j is a graph database NoSQL database that specializes in storing and querying graph data structures. It provides high-performance graph traversal, pattern matching, and graph analytics capabilities, making it ideal for applications such as social networks, recommendation engines, and network analysis.

4.1.5 SQL VS NoSQL

SQL (Structured Query Language) and NoSQL (Not Only SQL) databases differ primarily in their data models, scalability, and query languages. SQL databases, also known as relational databases, follow a structured data model based on tables with rows and columns. They enforce a rigid schema and provide ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data integrity and consistency for transactional processing. SQL databases use SQL for querying and manipulating data, providing a standardized syntax for expressing complex queries, joins, and transactions. They are well-suited for applications requiring structured data, complex transactions, and strict data integrity requirements, such as financial systems, ERP (Enterprise Resource Planning), and traditional OLTP (Online Transaction Processing) applications.

In contrast, NoSQL databases support various data models, including key-value, document, columnar, and graph databases, offering flexibility in data modeling and schema design. They are designed for horizontal scalability, enabling organizations to scale out across distributed clusters of commodity hardware. NoSQL databases may sacrifice some ACID properties in favor of scalability and performance, providing eventual consistency or relaxed consistency models. They use different query languages or APIs depending on the data model, such as JSON-like queries for document databases or CQL (Cassandra Query Language) for wide-column stores. NoSQL databases are suitable for handling large volumes of diverse and unstructured data, real-time data processing, and scalable web applications, including content management, social media analytics, IoT (Internet of Things), and real-time analytics.

Data Model:
SQL databases, also known as relational databases, follow a structured data model based on tables with rows and columns. They enforce a rigid schema that defines the structure of the data.
NoSQL databases, on the other hand, support various data models, including key-value, document, columnar, and graph databases. They offer flexibility in data modeling, allowing for schema-less or schema-flexible approaches.

Scaling:
SQL databases typically scale vertically by adding more resources (CPU, RAM) to a single server. Scaling beyond the capacity of a single server can be challenging and expensive.
NoSQL databases are designed for horizontal scalability, enabling organizations to scale out across distributed clusters of commodity hardware. They can handle large volumes of data and increasing workloads by simply adding more nodes to the cluster.

Transactions:
SQL databases provide ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data integrity and consistency for transactional processing.
NoSQL databases may sacrifice some ACID properties in favor of scalability and performance. They often provide eventual consistency or relaxed consistency models, which may be suitable for certain use cases but may not guarantee immediate consistency across distributed nodes.

Query Language:
SQL databases use Structured Query Language (SQL) for querying and manipulating data. SQL provides a standardized syntax for expressing complex queries, joins, and transactions.
NoSQL databases use various query languages or APIs depending on the data model. For example, MongoDB uses JSON-like queries for document databases, while Cassandra uses CQL (Cassandra Query Language) for wide-column stores.

Use Cases:
SQL databases are well-suited for applications requiring structured data, complex transactions, and strict data integrity requirements. Common use cases include financial systems, ERP (Enterprise Resource Planning), and traditional OLTP (Online Transaction Processing) applications.
NoSQL databases are suitable for handling large volumes of diverse and unstructured data, real-time data processing, and scalable web applications. Common use cases include content management, social media analytics, IoT (Internet of Things), and real-time analytics.

Schema Design:
SQL databases enforce a fixed schema, requiring predefined data mappings and schema migrations for schema changes. This can add complexity and overhead to application development.
NoSQL databases support dynamic schema design, allowing for flexible data structures and adaptation to changing data requirements without the need for schema migrations. This simplifies data modeling and application development, facilitating agile development and iteration.

4.2 HADOOP

Hadoop is an open-source distributed computing framework designed to store, process, and analyze large volumes of data across clusters of commodity hardware. Originally developed by Doug Cutting and Mike Cafarella in 2005, Hadoop has since become a cornerstone technology in the big data ecosystem, powering a wide range of applications and use cases in various industries.

At its core, Hadoop consists of two main components: the Hadoop Distributed File System (HDFS) for distributed storage and the MapReduce programming model for parallel processing. HDFS divides large datasets into smaller blocks and distributes them across multiple nodes in a cluster, providing fault tolerance and high availability. MapReduce, inspired by functional programming concepts, enables parallel processing of data by dividing tasks into smaller, independent units called mappers and reducers, which execute in parallel across nodes in the cluster.

Hadoop is an open-source software framework used for storing and processing Big Data in a distributed manner on large clusters of commodity hardware. Hadoop is licensed under Apache Software Foundation (ASF).

Hadoop is written in the Java programming language and ranks among the highest-level Apache projects. Doug Cutting and Mike J. Cafarella developed Hadoop. By getting inspiration from Google, Hadoop is using technologies like Map-Reduce programming model as well as Google file system (GFS).

It is optimized to handle massive quantities of data that could be structured, unstructured or semi-structured, using commodity hardware, that is, relatively inexpensive computers.
It is intended to work upon from a single server to thousands of machines each offering local computation and storage. It supports the large collection of data set in a distributed computing environment.

4.2.1 Features of Hadoop

Hadoop provides a diverse set of capabilities that make it a flexible and robust framework for storing, manipulating, and examining large volumes of data.
Notable characteristics of Hadoop encompass:

Distributed Storage: Hadoop Distributed File System (HDFS) distributes large datasets across clusters of commodity hardware, providing fault tolerance and high availability. It enables scalable storage of data across multiple nodes in the cluster.

Scalability: Hadoop is designed to scale horizontally, allowing organizations to handle growing volumes of data by adding more nodes to the cluster. This scalability ensures that Hadoop can efficiently process and analyze large datasets, regardless of size.

Parallel Processing: Hadoop uses the MapReduce programming model for parallel processing of data. It divides data processing tasks into smaller, independent units called mappers and reducers, which execute in parallel across nodes in the cluster. This parallelism enables Hadoop to process large volumes of data quickly and efficiently.

Fault Tolerance: Hadoop provides fault tolerance by replicating data across multiple nodes in the cluster. If a node fails or becomes unavailable, Hadoop can automatically redistribute tasks to other nodes, ensuring that data processing continues uninterrupted.

Cost-Effectiveness: Hadoop is built using open-source software and can run on commodity hardware, making it a cost-effective solution for storing and processing big data. Organizations can scale their Hadoop clusters as needed without incurring significant hardware or licensing costs.

Flexibility: Hadoop supports a variety of data types and formats, including structured, semi-structured, and unstructured data. It can process data in various formats, such as text, CSV, JSON, XML, and more, making it suitable for a wide range of use cases and applications.

Ecosystem: Hadoop has a rich ecosystem of tools and technologies that extend its capabilities for data storage, processing, and analysis. These include Apache Hive for SQL-like querying, Apache Pig for data processing, Apache Spark for in-memory processing, Apache HBase for real-time NoSQL database operations, Apache Kafka for real-time data streaming, and many others.

Security: Hadoop provides robust security features, including authentication, authorization, and encryption, to protect data stored and processed in the cluster. It supports integration with authentication mechanisms such as Kerberos and LDAP, as well as data encryption at rest and in transit.

4.2.2 Key Advantages of Hadoop

Scalability: Hadoop is designed to scale horizontally, allowing organizations to expand their data storage and processing capabilities simply by adding more commodity hardware to the cluster. This scalability ensures that Hadoop can handle growing volumes of data without sacrificing performance or reliability.

Cost-Effectiveness: Hadoop is built using open-source software and can run on commodity hardware, making it a cost-effective solution for storing, processing, and analyzing big data. Organizations can avoid the high costs associated with proprietary hardware and software licenses, while still achieving the scalability and performance required for their data needs.

Fault Tolerance: Hadoop provides built-in fault tolerance mechanisms to ensure data reliability and availability. By replicating data across multiple nodes in the cluster, Hadoop can continue to operate even if individual nodes fail or become unavailable. This fault tolerance helps minimize the risk of data loss and ensures uninterrupted data processing and analysis.

Flexibility: Hadoop supports a wide range of data types and formats, including structured, semi-structured, and unstructured data. It can handle diverse datasets, such as text, CSV, JSON, XML, and more, making it suitable for a variety of use cases and applications. Additionally, Hadoop's flexible data model allows organizations to adapt to changing data requirements and evolving business needs over time.

Parallel Processing: Hadoop uses the Map Reduce programming model for parallel processing of data, allowing tasks to be divided into smaller, independent units that can be executed in parallel across nodes in the cluster. This parallelism enables Hadoop to process large volumes of data quickly and efficiently, resulting in faster data analysis and insights.

Rich Ecosystem: Hadoop has a rich ecosystem of tools and technologies that extend its capabilities for data storage, processing, and analysis. These include Apache Hive for SQL-like querying, Apache Pig for data processing, Apache Spark for in-memory processing, Apache HBase for real-time NoSQL database operations, Apache Kafka for real-time data streaming, and many others. This ecosystem provides organizations with a wide range of options for building and deploying big data solutions tailored to their specific needs

.

4.2.3 Versions of Hadoop

Hadoop is an open-source software framework that enables the storage and processing of data in a distributed network, allowing for parallel data processing instead of relying on a centralized system. Hadoop's capabilities allow it to function as a highly dependable batch processing engine and a system for managing tiered storage and resources. As the complexity of the stored and processed data rises, Hadoop also evolves with multiple versions to solve concerns such as bug patches and to simplify the complicated data processes. The updates are immediately incorporated as Hadoop development adheres to the trunk-branch approach, where the base code is continuously updated and fixes are implemented in separate branches.
Hadoop has two versions:
a) Hadoop 1.x (Version 1) and
b) Hadoop 2 (Version 2)

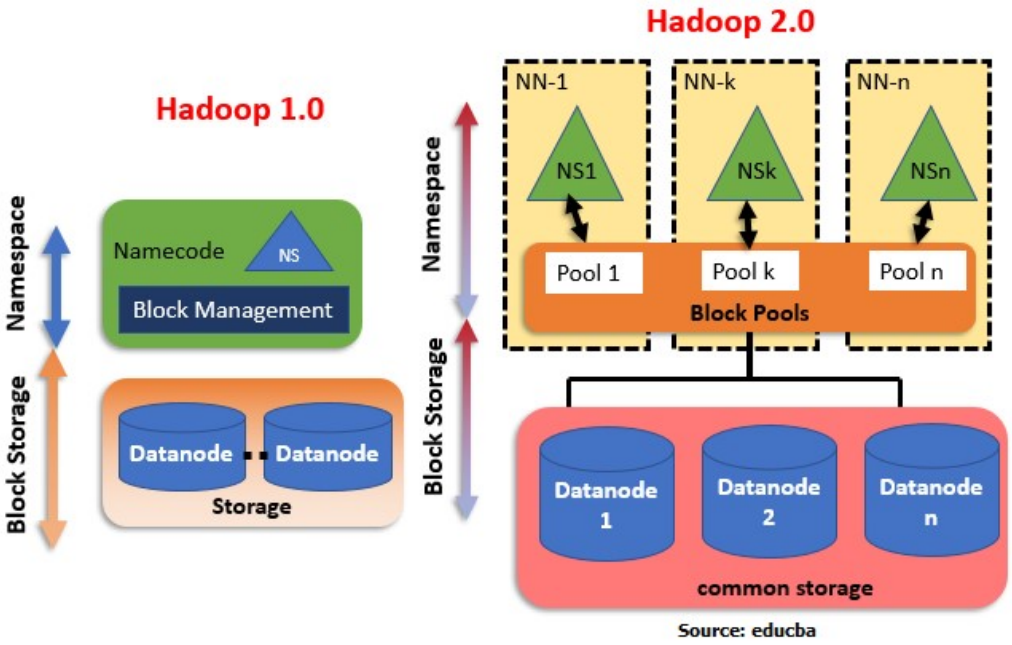


Figure 4.3 Hadoop Versions

Hadoop 0.1.0 (2006): This was the initial release of Hadoop, which included HDFS for distributed storage and Map Reduce for distributed processing. It was primarily used for experimental purposes.

Hadoop 0.20.0 (2009): This release marked significant improvements in stability, scalability, and performance. It introduced features such as support for append operations in HDFS and improvements in fault tolerance and job scheduling in Map Reduce.

Hadoop 1.0.0 (2011): This release marked the transition of Hadoop from the 0.x series to the 1.x series. It included several enhancements and stability improvements, making it suitable for production deployments. The primary components included HDFS, MapReduce, and Hadoop Common.

Hadoop 2.0.0 (2012): This release introduced major architectural changes, including the introduction of YARN (Yet Another Resource Negotiator) as a resource management layer. YARN decoupled resource management and job scheduling from Map Reduce, allowing Hadoop to support multiple processing frameworks. It also included HDFS High Availability (HA) and NameNode Federation for improved scalability and reliability.

Hadoop 2.x (2013-2019): The 2.x series saw several incremental releases with improvements in performance, scalability, and reliability. It introduced features such as HDFS snapshots, HDFS encryption, and enhancements to YARN for better resource management and job scheduling. It also saw the emergence of new processing frameworks like Apache Spark and Apache Flink alongside Map Reduce.

Hadoop 3.0.0 (2017): This release introduced significant enhancements and new features, including support for erasure coding in HDFS for storage efficiency, containerization support in YARN for better resource isolation, and improvements in Hadoop Common for better compatibility with modern hardware and software ecosystems.

Hadoop 3.x (2017-present): The 3.x series continues to receive incremental updates with improvements in performance, security, and usability. It includes features such as support for GPU acceleration, enhancements to HDFS for better scalability and reliability, and improvements to YARN for better resource utilization and management.

4.2.4 Overview of Hadoop Ecosystem

The Hadoop ecosystem refers to a collection of open-source software projects and tools that complement the core components of the Hadoop framework. The Hadoop ecosystem encompasses a wide range of technologies across various categories, including storage, processing, data ingestion, data governance, security, and analytics.

HDFS	Hadoop Distributed File System
	Yet Another Resource Negotiator
YARN	
	Data processing using programming
MapReduce	
	Inmemory Data Processing
Spark	
	Data Przocessing Services using Query (SQL-Like)
PIG, HIVE	
	NoSQL Database
HBase	
	Machine Learning

Mahout, Spark MLlib	SQL on Hadoop
Apache Drill	Managing Cluster
Zookeeper	Job Scheduling
Oozie	Data Ingesting Services
Flume, Sqoop	Searching & Indexing
Solr&Lucene	Provision, Monitor and Maintain cluster
Ambari	

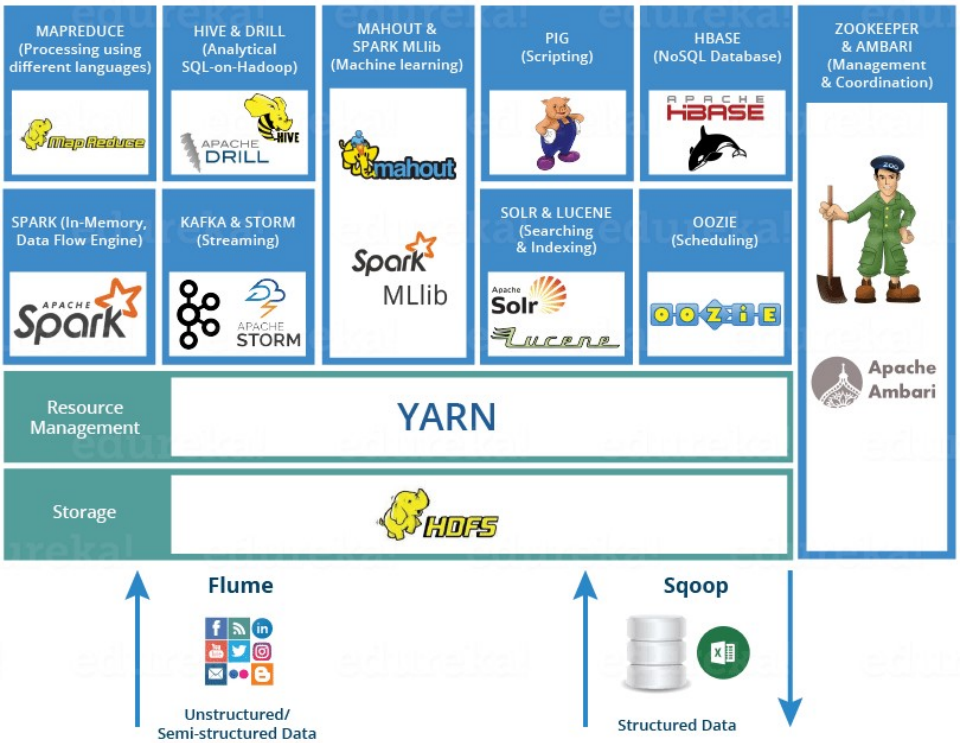


Figure 4.4 Hadoop Eco System

4.2.4.1 HDFS

Hadoop Distributed File System is the core component or you can say, the backbone of Hadoop Ecosystem.

HDFS is the one, which makes it possible to store different types of large data sets (i.e. structured, unstructured and semi structured data).

HDFS creates a level of abstraction over the resources, from where we can see the whole HDFS as a single unit.

It helps us in storing our data across various nodes and maintaining the log file about the stored data (metadata).
HDFS has two core components, i.e. **Name Node and Data Node**.

The **Name Node** is the main node and it doesn't store the actual data. It contains metadata, just like a log file or you can say as a table of content. Therefore, it requires less storage and high computational resources.

On the other hand, all your data is stored on the **Data Nodes** and hence it requires more storage resources. These Data Nodes are commodity hardware (like your laptops and desktops) in the distributed environment. That's the reason, why Hadoop solutions are very cost effective.

You always communicate to the Name Node while writing the data. Then, it internally sends a request to the client to store and replicate data on various Data Nodes.

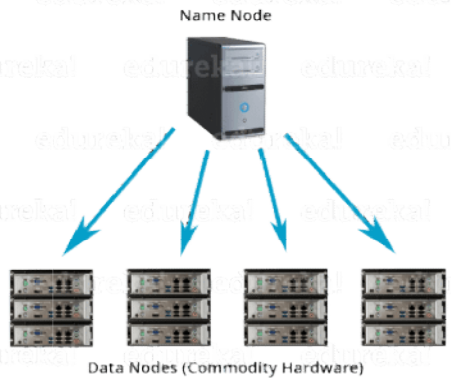


Figure 4.4 HDFS nodes

4.2.4.2 YARN

Consider YARN as the brain of your Hadoop Ecosystem. It performs all your processing activities by allocating resources and scheduling tasks.

It has two major components, i.e. **Resource Manager and Node Manager**.

Resource Manager is again a main node in the processing department.
It receives the processing requests, and then passes the parts of requests to corresponding Node Managers accordingly, where the actual processing takes place.

Node Managers are installed on every Data Node. It is responsible for execution of task on every single Data Node.

Schedulers: Based on your application resource requirements, Schedulers perform scheduling algorithms and allocates the resources.

Applications Manager: While Applications Manager accepts the job submission, negotiates to containers (i.e. the Data node environment where process executes) for executing the application specific Application Master and monitoring the progress. Application Masters are the deamons which reside on Data Node and communicates to containers for execution of tasks on each Data Node. Resource Manager has two components, i.e. **Schedulers** and **Applications Manager**.

4.2.4.3. MAPREDUCE

It is the core component of processing in a Hadoop Ecosystem as it provides the logic of processing. In other words, Map Reduce is a software framework which helps in writing applications that processes large data sets using distributed and parallel algorithms inside Hadoop environment.

In a Map Reduce program, **Map() and Reduce()** are two functions.

The **Map function** performs actions like filtering, grouping and sorting.
While **Reduce function** aggregates and summarizes the result produced by map function.
The result generated by the Map function is a key value pair (K, V) which acts as the input for Reduce function.

Student	Department	Count	(Key, Value), Pair
Student 1	D1	1	(D1, 1)
Student 2	D1	1	(D1, 1)
Student 3	D1	1	(D1, 1)
Student 4	D2	1	(D2, 1)
Student 5	D2	1	(D2, 1)
Student 6	D3	1	(D3, 1)
Student 7	D3	1	(D3, 1)

Figure 4.5 output from the map function

Let us take the above example to have a better understanding of a MapReduce program.
We have a sample case of students and their respective departments. We want to calculate the number of students in each department. Initially, Map program will execute and calculate the students appearing in each department, producing the key value pair as mentioned above. This key value pair is the input to the Reduce function. The Reduce function will then aggregate each department and calculate the total number of students in each department and produce the given result.

Department	Total Student
D1	3
D2	2
D3	2

Figure 4.6 output from the reduce function

4.2.4.4 APACHE PIG

PIG has two parts: **Pig Latin**, the language and **the pig runtime,** for the execution environment. You can better understand it as Java and JVM.
It supports *pig latin* language, which has SQL like command structure.
As everyone does not belong from a programming background. So, Apache PIG relieves them. *You might be curious to know how?*
Well, I will tell you an interesting fact:

10 line of pig latin = approx. 200 lines of Map-Reduce Java code
But don't be shocked when I say that at the back end of Pig job, a map-reduce job executes.
The compiler internally converts pig latin to Map Reduce. It produces a sequential set of Map Reduce jobs, and that's an abstraction (which works like black box).
PIG was initially developed by Yahoo.
It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analyzing huge data sets.
How Pig works?
In PIG, first the load command, loads the data. Then we perform various functions on it like grouping, filtering, joining, sorting, etc. At last, either you can dump the data on the screen or you can store the result back in HDFS.

4.2.4.5 APACHE HIVE
Facebook created HIVE for people who are fluent with SQL. Thus, HIVE makes them feel at home while working in a Hadoop Ecosystem.

Basically, HIVE is a data warehousing component which performs reading, writing and managing large data sets in a distributed environment using SQL-like interface.
HIVE + SQL = HQL

The query language of Hive is called Hive Query Language(HQL), which is very similar like SQL.
It has 2 basic components: Hive Command Line and JDBC/ODBC driver.
The **Hive Command line** interface is used to execute HQL commands.
While, Java Database Connectivity (**JDBC**) and Object Database Connectivity (**ODBC**) is used to establish connection from data storage.

Secondly, Hive is highly scalable. As, it can serve both the purposes, i.e. large data set processing (i.e. Batch query processing) and real time processing (i.e. Interactive query processing).
It supports all primitive data types of SQL.

You can use predefined functions, or write tailored user defined functions (UDF) also to accomplish your specific needs.

4.2.4.6 APACHE MAHOUT
Now, let us talk about Mahout which is renowned for machine learning. Mahout provides an environment for creating machine learning applications which are scalable.

Machine learning algorithms allow us to build self-learning machines that evolve by itself without being explicitly programmed. Based on user behavior, data patterns and past experiences it makes important future decisions. You can call it a descendant of Artificial Intelligence (AI).
What Mahout does?

It performs collaborative filtering, clustering and classification. Some people also consider frequent item set missing as Mahout's function. Let us understand them individually:

Collaborative filtering: Mahout mines user behaviors, their patterns and their characteristics and based on that it predicts and make recommendations to the users. The typical use case is E-commerce website.

Clustering: It organizes a similar group of data together like articles can contain blogs, news, research papers etc.
Classification: It means classifying and categorizing data into various sub-departments like articles can be categorized into blogs, news, essay, research papers and other categories.

Frequent item set missing: Here Mahout checks, which objects are likely to be appearing together and make suggestions, if they are missing. For example, cell phone and cover are brought together in general. So, if you search for a cell phone, it will also recommend you the cover and cases.
Mahout provides a command line to invoke various algorithms. It has a predefined set of library which already contains different inbuilt algorithms for different use cases.

4.2.4.7 APACHE SPARK

Apache Spark is a framework for real time data analytics in a distributed computing environment. The Spark is written in Scala and was originally developed at the University of California, Berkeley.

It executes in-memory computations to increase speed of data processing over Map-Reduce. It is 100x faster than Hadoop for large scale data processing by exploiting in-memory computations and other optimizations. Therefore, it requires high processing power than Map-Reduce.

Spark comes packed with high-level libraries, including support for R, SQL, Python, Scala, Java etc. These standard libraries increase the seamless integrations in complex workflow. Over this, it also allows various sets of services to integrate with it like MLlib, GraphX, SQL + Data Frames, Streaming services etc. to increase its capabilities.

4.2.4.8 APACHE HBASE

HBase is an open source, non-relational distributed database. In other words, it is a NoSQL database.

It supports all types of data and that is why, it's capable of handling anything and everything inside a Hadoop ecosystem.It is modelled after Google's BigTable, which is a distributed storage system designed to cope up with large data sets.

The HBase was designed to run on top of HDFS and provides BigTable like capabilities.
It gives us a fault tolerant way of storing sparse data, which is common in most Big Data use cases.The HBase is written in Java, whereas HBase applications can be written in REST, Avro and Thrift APIs.

For better understanding, let us take an example. You have billions of customer emails and you need to find out the number of customers who has used the word complaint in their

emails. The request needs to be processed quickly (i.e. at real time). So, here we are handling a large data set while retrieving a small amount of data. For solving these kind of problems, HBase was designed.

4.2.4.9 APACHE DRILL

As the name suggests, Apache Drill is used to drill into any kind of data. It's an open source application which works with distributed environment to analyze large data sets.
It is a replica of Google Dremel.
It supports different kinds NoSQL databases and file systems, which is a powerful feature of Drill. For example: Azure Blob Storage, Google Cloud Storage, HBase, MongoDB, MapR-DB HDFS, MapR-FS, Amazon S3, Swift, NAS and local files.

So, basically the main aim behind Apache Drill is to provide scalability so that we can process petabytes and exabytes of data efficiently (or you can say in minutes).
The main power of Apache Drill lies in ***combining a variety of data stores just by using a single query.***

Apache Drill basically follows the ANSI SQL.
It has a powerful scalability factor in supporting millions of users and serve their query requests over large scale data.

4.2.4.10 APACHE ZOOKEEPER

Apache Zookeeper is the coordinator of any Hadoop job which includes a combination of various services in a Hadoop Ecosystem.
Apache Zookeeper coordinates with various services in a distributed environment.

Before Zookeeper, it was very difficult and time consuming to coordinate between different services in Hadoop Ecosystem. The services earlier had many problems with interactions like common configuration while synchronizing data. Even if the services are configured, changes in the configurations of the services make it complex and difficult to handle. The grouping and naming was also a time-consuming factor.

Due to the above problems, Zookeeper was introduced. It saves a lot of time by performing synchronization, configuration maintenance, grouping and naming.

4.2.5 Hadoop Distribution

Hadoop is a versatile technology solution that is open-source and offers exceptional scalability. It provides low-cost storage systems and enables fast-paced big data analytics. Additionally, it helps reduce server expenses.

Hadoop vendor distributions address the limitations and problems associated with the open source version of Hadoop. These releases offer additional features that specifically target:

Support
The majority of Hadoop suppliers offer technical coaching and help to facilitate the adoption of Hadoop for enterprise-level workloads and mission-critical applications.

Reliability:
Hadoop suppliers respond rapidly whenever an issue is identified. In order to enhance the stability of commercial systems, updates and fixes are promptly implemented.

Completeness:
Hadoop suppliers augment their distributions with a range of supplementary tools that enable clients to tailor the Hadoop application to suit their individual requirements.

Fault Tolerance:
Due to the default replication factor of three, the data is highly available and capable of withstanding faults. This makes commercial solutions more stable, as updates and changes are promptly provided.

4.2.6 Hadoop versus SQL

Data Processing
Data processing refers to the conversion and manipulation of raw data into a more meaningful and useful format.

The SQL processing capacity is insufficient for the volume of data. It performs exceptionally well with Gigabytes. However, when dealing with a substantial volume of data, such as processing Terabytes or Petabytes, SQL is unable to match the anticipated requirements. Hadoop is specifically built to effectively manage large volumes of diverse data commonly found in modern companies. For large-scale enterprises, Hadoop is the most suitable option.

Processing Speed
Hadoop comprises two fundamental components. The HDFS, also known as the Hadoop Distributed File System, utilises the Map Reduce framework and the Flat File System to handle data processing. It lacks the capability to handle data in real-time or support OLTP (Online Transaction Processing). It facilitates the efficient handling of vast amounts of data, a critical component in the field of data mining. OLAP facilitates the execution of complex tasks by utilising large-scale processing and performing aggregations. The processing speed varies depending on the data set you input. The duration can range from a few minutes to several hours.

SQL enables real-time data processing, also known as online transaction processing (OLTP). SQL has faster processing performance due to its ability to handle normalised data, but it lacks capabilities for batch processing.

ACID property
It refers to a set of characteristics that ensure reliability and consistency in database transactions. The acronym ACID stands for Atomicity, Consistency, Isolation, and Durability.

SQL provides support for the ACID properties of RDBMS, including Consistency, Isolation, Atomicity, and Durability. Conversely, in Hadoop, it is necessary to write code for all situations in order to do this. Undoubtedly, SQL is surpassing Hadoop in this regard.

Fault Tolerance
Fault tolerance refers to the ability of a system to continue functioning properly even in the presence of faults or failures.

In traditional Relational Database Management Systems (RDBMS), the process of recovering lost data due to network problems or corruption incurs significant expenses, time, and resource utilization. This is a case involving SQL. However, Hadoop employs a technique in which data is replicated across three distinct layers. Although it may appear to be a waste of effort, if the data stored in one node is lost, it may be quickly retrieved from the other data nodes.

Cost

The primary distinction between these two is that acquiring a SQL server requires a higher financial investment. The server incurs significant expenses, and in the event of storage depletion, more fees must be paid to acquire the necessary storage. However, Hadoop is a freely available open-source platform, meaning it does not require any payment. Additionally, for handling larger-scale data processing, Hadoop is considered the most optimal choice. This is why many businesses are choosing to use Hadoop.

Architecture

Hadoop possesses a flexible structure and has the capability to store and handle vast quantities of data. The system has the capability to store films, photos, sensor data, and various other sorts of data in real-time.

Conversely, SQL has a fixed schema and is limited to storing data in a tabular or organised fashion.

Functional programming

Hadoop enables programming in languages such as Scala, Java, and Python. To obtain any additional functionality, you can acquire it by registering User-defined Functions or UDF in the HDFS. However, with a Relational Database Management System (RDBMS), you are not allowed to write User-Defined Functions (UDFs), which leads to a rise in the complexity of SQL. In addition, the data stored in Hadoop can be readily accessed via Pig, Hive, Sqoop, and other related ecosystems.

Other differences

Hadoop utilises the Hadoop Distributed File System (HDFS) to store data and employs the MapReduce technology to process the data, resulting in enhanced optimisation. However, SQL lacks any optimisation strategies.

In SQL, data updates require numerous read and write operations, however in Hadoop, data just needs to be written once and can be read multiple times.

SQL utilises proprietary hardware, whereas Hadoop utilises commodity gear. Upon observing the numerous disparities among the prevalent data management platforms, it becomes apparent that Hadoop is the optimal selection. To acquire expertise in Hadoop, you can enhance your skills by enrolling in one of the top-rated online training courses available.

4.3 SUMMARY

This chapter delves into the expansive landscape of big data technologies and ecosystems, elucidating the paradigm shift in data management ushered in by the proliferation of massive datasets. It navigates through the intricate web of tools and frameworks designed to tackle the

challenges posed by big data, highlighting the emergence of NoSQL databases as a scalable and flexible alternative to traditional relational databases. Furthermore, the chapter explores the foundational role of Hadoop in enabling distributed storage and processing of vast amounts of data across commodity hardware clusters, thus empowering organizations to harness the potential of big data for insights and innovation. Through a comprehensive examination of these pivotal technologies, the chapter underscores their collective significance in driving the evolution of modern data ecosystems towards greater efficiency and agility.

4.4 TECHNICAL TERMS

BigData, NoSQL, Hadoop, Hadoop EcoSystem.

4.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Provide examples of industries or domains where Big Data technologies have revolutionized traditional data management and analysis practices.
2. Compare and contrast traditional relational database management systems (RDBMS) with NoSQL databases. Discuss the advantages and disadvantages of each approach
3. Describe the Hadoop ecosystem, including its core components and their respective roles in distributed data storage and processing

Short Questions:

1. What is the primary objective of Hadoop in the context of big data processing?
2. Name two popular components of the Hadoop ecosystem and briefly explain their roles.
3. What distinguishes NoSQL databases from traditional relational databases?
4. Provide one example of a use case where NoSQL databases excel over traditional SQL databases.
5. How does the Hadoop ecosystem facilitate the processing and analysis of large-scale datasets?

4.9 SUGGESTED READINGS

1. Seema Acharya, SubhashiniChellappan --- Big Data And Analytics secondedition, Wiley
2. Seema Acharya--Data Analytics using R, McGraw Hill education (India) Private Limited.
3. Big Data Analytics, Introduction to Hadoop, Spark, and Machine-Learning, Rajkamal, PreetiSaxena, McGraw Hill, 2018.
4. Big Data, Big Analytics: Emerging Business intelligence and Analytic trends forToday's Business, Michael Minelli, Michelle Chambers, and AmbigaDhiraj, John Wiley &Sons, 2013

AUTHOR: **Dr. B. Reddaiah**

LESSON- 5
INTRODUCTION TO R

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the importance of data
- students should have a solid understanding of the fundamentals of R programming
- Understand R Programming purpose, advantages, data types, coercion, working with variables, and managing objects in the R workspace.
- Equipped with the necessary knowledge to begin writing and executing basic R scripts for data analysis and statistical computing tasks.

STRUCTURE OF THE LESSION:

5.1 WHAT AND WHY IS R

R is a programming language and software environment that assists in the analysis of statistical data, the representation of images, and the generation of reports. R is a programming language that was initially developed by Ross Ihaka and Robert Gentleman at the University of Auckland in New Zealand. The R Development Core Team is currently expanding the capabilities of R.

The core of R is an interpreted computer language that enables modular programming through the use of functions, as well as branching and looping. R makes it possible to integrate with procedures written in languages such as C, C++,.Net, Python, or FORTRAN, which leads to increased efficiency.

The GNU General Public Licence makes R freely accessible to the public, and pre-compiled binary copies of the programme are made available for a variety of operating systems, including Linux, Windows, and Mac Operating Systems.

R is a piece of free software that is shared under a copy left licence similar to that of GNU. It is also an official component of the GNU project known as GNU S.

Starting from mid-1997, a central organisation known as the "R Core Team" has had the authority to make changes to the R source code archive.

5.1.1 Features of R

Open-source:R is an open-source programming language, which means it is freely available for anyone to download, use, and modify.

Statistical Computing and Graphics:R was initially developed for statistical computing and graphics, making it a powerful tool for data analysis, visualization, and modeling.

Extensive Libraries (CRAN):R boasts a vast ecosystem of packages and libraries contributed by the community, available through the Comprehensive R Archive Network (CRAN).These packages cover a wide range of functionalities, including data manipulation, statistical modeling, machine learning, visualization, and more.

Data Handling: R provides efficient data handling capabilities, allowing users to import, manipulate, clean, and transform data from various sources, including files, databases, and web APIs.

Vectorized Operations: R is designed for vectorized operations, which means it can perform computations on entire vectors or matrices without the need for explicit looping. This feature enhances code efficiency and readability, particularly in statistical and numerical computations.

Statistical Analysis: R offers a comprehensive suite of statistical functions and methods for descriptive statistics, hypothesis testing, regression analysis, time series analysis, clustering, and more.

Graphics and Visualization: R provides extensive capabilities for creating high-quality static and interactive visualizations. Packages like ggplot2, lattice, and plotly offer flexible and customizable plotting functionalities for exploring and presenting data graphically.

5.1.2 Applications of R

Why would you use R programming in the real world?

1. Statistical analysis and making sense of data: At its core, R is the same thing as statistical research. It comes with all the tools you need to do a wide range of statistical tests, from simple descriptive statistics to complex regression models. R is great for more than just numbers. It's also great for showing data visually. ggplot2 and other similar packages make it easy to make interesting graphs and charts that help you understand large datasets visually.

2. Exploring and cleaning the data: Exploring and cleaning the data are the first steps in any data analysis process. Because of what it can do, R is a great choice for dealing with missing values and outliers and checking the quality of the data as a whole before doing more in-depth analysis. In real life, R's powerful data preparation tools make sure that datasets are carefully prepared and improved so that insights are correct and reliable.

3. Predictive Modelling and Machine Learning: R has a lot of features for both predictive modelling and machine learning. It has many methods for regression, classification, and clustering, which makes it a great language for making models that can predict the future. R's machine learning features are very useful for real-time tasks like predicting stock prices, customer behaviour, or disease results, as they help make decisions based on data.

4. Biostatistics and Healthcare: R is a key tool in biostatistics; it is used to look at data from clinical trials, do epidemiological studies, and help healthcare workers make decisions based on data. Some of the ways it can be used in healthcare are in genomics, where it is very helpful for looking at genetic data, finding trends linked to diseases, and making personalised medicine possible.

5. Finance and Risk Management: Risk modelling, portfolio optimisation, and analysing market trends are all things that the financial industry does with R. In financial analytics, where real-time insights can drive strategic decisions, R's ability to work with big datasets is very important.

People who want to become data scientists can learn how to use R programming for financial research and risk management by taking a well-rounded Data Science course.

6. Social Sciences and Market Research: R is used a lot in the social sciences to look at survey results, social media sentiment, and general opinion. Because it is so flexible, researchers can use it to learn from very large and different social datasets.

7. Environmental Science and Climate Research: R makes a big difference in environmental science by looking at climate data, guessing what will happen to the environment, and figuring out how our actions affect environments. Its uses in climate studies are very important for understanding and solving problems related to the environment. As worries about the planet's future grow, data scientists are turning to R for environmental study and climate modelling. This shows how R can be used in the real world to solve problems.

5.2 ADVANTAGES OF R OVER OTHER PROGRAMMING LANGUAGES

R offers several advantages over other programming languages, particularly in the realm of statistical analysis and data visualization. Its specialized focus on statistics means that it comes equipped with a wide range of libraries and packages tailored specifically for these tasks. This makes R particularly efficient and effective for data manipulation, modeling, and generating insightful visualizations.

Additionally, its open-source nature fosters a collaborative community of developers who continuously contribute new packages and provide support, ensuring that R remains cutting-edge and adaptable to evolving needs. Its interoperability with other languages allows for seamless integration into existing workflows, further enhancing its utility. Overall, R's specialized features, robust community support, and interoperability make it a preferred choice for statisticians, data scientists, and researchers tackling complex data analysis tasks

Table 5.1 shows that how the R language is differentiate from python and Java

R Programming	Python	Java
It was stably released in 2014	it was stably released in 1996	It was stably released in 1995
It has more functions and packages	it has less functions and packages	it has large number of inbuilt functions and packages
it is an interpreter base language	it is an interpreter base language	it is interpreter and compiled based language
It is statistical design and graphics programming language	It is easy to understand.	It is easy to learn and understand
It is difficult to learn and understand	It is easy to understand	It is easy to lean and understand
R ismostly use for data analysis	Generic programming tasks like a design of software	Java is mostly used in design of windows applications and web applications

5.3 DATA TYPES IN R

The data type of the R object that is assigned to a variable is the one that is assigned to the variable itself in R. A variable is not declared with any data type. Therefore, R is referred to as a dynamically typed language, which indicates that when we use it in a programme, we have the ability to alter the data type of the same variable several times with each new instance.

It is the responsibility of the Data Types to determine the sort of value that a variable possesses and the kinds of mathematical, relational, or logical operations that can be performed on it without resulting in an error.

5.3.1 Vectors

Vectors are the most basic R data objects and there are six types of atomic vectors. Below are the six atomic vectors:

Logical: It is used to store logical value like TRUE or FALSE.

Numeric: It is used to store both positive and negative numbers including real number.

Eg: 34, 3.5271 , 85138

Integer: It holds all the integer values i.e. all the positive and negative whole numbers.

Eg: 73278,91, -5623.7542 , 0

Complex: These are of the form x + yi, where x and y are numeric and i represents the square root of -1.

Eg: 8+5i

Character: It is used to store either a single character, group of characters(words) or a group of words together. The characters may be defined in either single quotes or double quotes.

Eg: "Rohit Arya", 'Acharya Nagarjuna University'.

In general, a vector is defined and initialized in the following manner:

```
vect =  m(6, 1, 48 , 72)
```

Or

```
vect<- c(6, 1, 48, 72)
```

5.3.2 List

Lists are quite similar to vectors, but Lists are the R objects which can contain elements of different types like − numbers, strings, vectors and another list inside it.

Eg:

```
vect<- c("ANU", 'B.Com','Lakshmi Shourya')
mylist <- list(vect, 66.3, 257192, TRUE)
mylist
Output:
[[1]]
[1] "ANU", 'B.Com','Lakshmi Shourya'
[[2]]
[1] 66.3
[[3]]
[1] 257192
[[4]]
[1] TRUE
```

5.3.3 Matrix

Matrix is the R object in which the elements are arranged in a two-dimensional rectangular layout.

The basic syntax for creating a matrix in R is −

matrix(data, nrow, ncol, byrow, dimnames)

Where:

data is the input vector which becomes the data elements of the matrix.

nrow is the number of rows to be created.

ncol is the number of columns to be created.

byrow is a logical clue. If TRUE, then the input vector elements are arranged by row.

dimname is the names assigned to the rows and columns.

Example:

```
Mymatrix <- matrix(c(1:25), nrow = 5,  ncol = 5, byrow = TRUE)
Mymatrix
```

Output:

```
[,1]   [,2]   [,3]   [,4]   [,5]
[1,]      1      2       3   4       5
[2,]      6      7       8     9     10
[3,]      11     12     13    14     15
[4,]      16     17     18    19     20
[5,]      21     22     23    24     25
```

5.3.4 Array

Arrays in R are data objects which can be used to store data in more than two dimensions. It takes vectors as input and uses the values in the dim parameter to create an array.

The basic syntax for creating an array in R is −

```
array(data, dim, dimnames)
```

Where:

data is the input vector which becomes the data elements of the array.

dim is the dimension of the array, where you pass the number of rows, column and the number of matrices to be created by mentioned dimensions.

dimname is the names assigned to the rows and columns.

Example:

```
Myarray <- array( c(1:16), dim=(4,4,2))
Myarray
```

Output:

```
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    5     9   13
[2,]    2    6    10   14
[3,]    3    7    11   15
[4,]    4    8    12   16

, , 2
     [,1] [,2] [,3] [,4]
[1,]    1    5    9    13
[2,]    2    6    10   14
[3,]    3    7    11   15
[4,]    4    8  12   16
```

5.3.5 Data Frame

A Data Frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values for each column. Below are some of the characteristics of a Data Frame that needs to be considered every time we work with them:

The column names should be non-empty.

Each column should contain the same amount of data items.

The data stored in a data frame can be of numeric, factor or character type.

The row names should be unique.

Example:

```
emp_id = c(201:205)
emp_name = c("Surya","Sri","Arya","Nivas","Reddaiah")
dept = c("CSE","EEE","ECE","Mechanical","Civil")
emp.data <- data.frame(emp_id, emp_name, dept)
emp.data

Output:

   emp_id    emp_name    dept
1 201         Surya        CSE
2 202         SriEEE
3 203         Arya         ECE
4 204          Nivas       Mechanical
5 205         Reddaiah     Civil
```

5.4 COERSION

Coercion includes type conversion. Type conversion means change of one type of data into another type of data. We have to type of coercion occurs:

1. Implicit Coercion
2. Explicit Coercion

5.4.1 Explicit Coercion:

In explicit coercion , we can change one data type to another data type by applying function. We create an object "x" which stores integer values from 1 to 6.

```
x<-0:6
```

We can check data type of "x" object.

```
class(x)
```

We used as.numeric() to change integer data type to numeric data type.

```
z<-as.numeric(x)
```

We check data type of z. It shows "numeric" data type.

```
class(z)
```

```
> x<-0:6
> class(x)
[1] "integer"
> z<-as.numeric(x)
> class(z)
[1] "numeric"
```

We can also change character data to numeric data as:

```
> v<-c("1","2")
> v
[1] "1" "2"
```

```
> as.numeric(v)
[1] 1 2
```

We also changed logical data to character data.

```
> x<-c(T,F)
> x
[1]  TRUE FALSE
> y<-as.character(x)
> y
[1] "TRUE"  "FALSE"
```

We also changed integer data to Logical data as:

```
> x<-0:6
> x
[1] 0 1 2 3 4 5 6
> f<-as.logical(x)
> f
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

When we changed numeric or integer to logical data , it will store 0 as FALSE and other values as TRUE. You can see here also .

```
> x<-c(-1,2,0)
> x
[1] -1  2  0
> class(x)
[1] "numeric"
> f<-as.logical(x)
> f
[1]  TRUE  TRUE FALSE
```

Some exceptions of explicit coercion :-

```
> x<-c("a","b","c")
> v<-as.numeric(x)
Warning message:
NAs introduced by coercion
> v
[1] NA NA NA
```

We are converting character data type to numeric data type. It will show NA . It will show missing in out object. It will not change character data to numeric because it includes values "a" which cannot be changed to numeric data.

5.4.2 Implicit Coercion :

When type conversion occurs by itself in R.

We input numeric and character data in an object . R converts numeric data to character data by itself.

```
> x<-c(1.7,"a")
> x
[1] "1.7" "a"
```

We input logical and numeric data in an object . Logical data convert to numeric data implicitly.

```
> y<-c(TRUE ,5)
> y
[1] 1 5
```

5.5 ls() COMMAND

In R, the `ls()` function is used to list the objects (variables, functions, etc.) that are currently stored in the workspace. When you call `ls()`, it will return a character vector containing the names of all the objects present in the current environment.

Variables whose names begin with a dot are, by default, not returned.

Syntax:

```
ls(name, pos = -1L, envir = as.environment(pos),

all.names = FALSE, pattern, sorted = TRUE)
```

where

name: This is the environment used in listing the available objects.

pos: This is an alternative argument to the parameter name for specifying the environment as a position in the search list.

envir: This is another alternative argument to name for specifying the environment.

all.names: This takes a logical value (TRUE or FALSE) indicating whether all the object names are returned (if TRUE) or the object names beginning with a . are omitted (if FALSE.)

pattern: Only names that match the pattern are returned.

sorted: This takes a logical value (TRUE or FALSE) indicating if the output character should be sorted alphabetically or not.

Example:

```
# creating R variables and functions
a <- 36
b <- sqrt(a)
c <-  a*b
.hide <- "ls will not show this if all.names= FALSE"

# implementing the ls() function by default
ls()
[1] "a" "b" "c" "r"

# implementing the ls() function to omit variables begining with dot(.)
ls(all.names='FALSE')
[1] "a" "b" "c" "r"
# implementing the ls() function to also return variables begining with (.)
ls(all.names='TRUE')
 [1] ".hide" "a" "b" "c" "r"
```

5.6 EXPRESSIONS

In R programming, expressions are combinations of variables, operators, functions, and other elements that produce a value when evaluated. Expressions can be simple, such as a single variable, or complex, involving multiple operations. Here are some examples of expressions in R:

1. Arithmetic expressions:

```
x <- 5
y <- 3
z <- x + y
```

In this example, `x + y` is an arithmetic expression that adds the values of `x` and `y` and assigns the result to `z`.

2. Function calls:

```
sqrt(16)
```

Here, `sqrt()` is a function call expression that computes the square root of the number `16`.

3. Logical expressions:

```
a <- 10
b <- 5
a > b
```

The expression `a > b` evaluates to `TRUE` because `a` is greater than `b`.

4. Conditional expressions:

```
if (a > b) {
  print("a is greater than b")
} else {
  print("a is not greater than b")
}
```

The expression `a > b` is a condition that determines which branch of the if-else statement is executed.

5. String concatenation:

```
string1 <- "Hello"
string2 <- "world"
paste(string1, string2)
```

The expression `paste(string1, string2)` concatenates `string1` and `string2` together.

6. Vector operations:

```
vector <- c(1, 2, 3, 4, 5)
sum(vector)
```

The expression `sum(vector)` calculates the sum of all elements in the vector.

These are just a few examples of expressions in R. In general, an expression can consist of any combination of R objects, operators, and function calls that produce a value.

5.8 VARIABLES

A variable, as explained in the preceding section, reserves a memory location and stores values that can be changed. A suitable variable name must be comprised of alphanumeric characters, as well as dots or underscores.

In R, variables are used to store data values. Variable names can consist of letters, numbers, periods, and underscores, but they cannot start with a number or a period followed by a number

Rules and conventions

In R, variable names must follow certain rules and conventions. Here are the key rules for naming variables in R:

1. Start with a letter or a period: Variable names must begin with a letter (either uppercase or lowercase) or a period. However, it's recommended to avoid starting variable names with a period as it has special meanings in R.
2. Subsequent characters: After the initial letter or period, variable names can contain letters, numbers, periods, and underscores.
3. Case sensitivity: R is case-sensitive, meaning uppercase and lowercase letters are considered distinct. For example, `myVariable`, `MyVariable`, and `MYVARIABLE` are all considered different variable names.
4. Reserved words: Avoid using reserved words as variable names. These are words that have special meaning in R, such as `if`, `else`, `function`, `for`, `while`, etc.
5. Special characters: While R allows the use of special characters like `+`, `-`, `*`, `/`, `%`, etc., in variable names, it's generally not recommended for clarity and ease of use.
6. Length: Variable names can be of any length, but it's good practice to keep them concise and meaningful.

Variables Assignment: Variables can be assigned in multiple ways

```
Assignment (=): var1 = "ANU"

Left (← ): var2 ← ","

Right (→): "R Programmng" → var3
```

Variable Name	Valid	Descrition
stu_marks1.	Valid	Contains letters, number, dot and
Underscore		
1stu_marks	Invalid	Starting with a number
Stu_marks@	Invalid	Has special character (@). Only dot
and underscore is allowed.		
.stu_marks, var.name	Valid	Can start with a dot, which is followed by an alphabet.
_stu_marks	Invalid	Should not start with underscore.
.2stu_marks	Invalid	Dot is followed by a number and hence invalid

5.9 SUMMARY

The chapter covers essential aspects of R programming, starting with an exploration of what R is and its significance. R stands out for its robust statistical and graphical capabilities, making it a preferred choice for data analysis and visualization tasks. It offers several advantages over other programming languages, such as its extensive collection of packages, active community support, and open-source nature. Understanding data types in R, including numeric, character, logical, and more, is crucial for effective data manipulation. Coercion is discussed as a process of converting data types in R, enabling seamless operations across different types. The ls() command is introduced for listing objects in the workspace, aiding in managing variables and expressions effectively. Overall, mastering expressions, variables,

and other fundamentals of R lays a solid foundation for proficient data analysis and programming in R.

5.10 TECHNICAL TERMS

Expressions, Variables, Python, Java

5.11 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the significance of R in the field of data science and statistical analysis.
2. Describe the various data types available in R and their significance in data analysis.
3. Discuss the concept of coercion in R. Provide examples to illustrate how coercion affects data types and operations in R programming.
4. Discuss the role of variables in R programming. How are variables defined, assigned, and utilized in data analysis tasks within the R environment?

Short Notes:

1. Compare and contrast the advantages of R over other programming languages
2. Explain the functionality and purpose of the `ls()` command in R.
3. Define expressions in the context of R programming.

5.12 SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. Crawley, M. J. (2012). The R book. John Wiley & Sons.
3. Albert, J. & Rizzo, M. (2012). R by Example. Springer
4. Teetor, P. (2011). R Cookbook. O'REILLY
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R.Cambridge.

AUTHOR: **Mr. G V Suresh**

LESSON- 6
CONTROL STRUCTURES

OBJECTIVES:

After going through this lesson, you will be able to

- Write if-else statements to make decisions based on conditions
- Understand the purpose and syntax of loops, including for and while loops.
- Apply control structures to manipulate and analyze data efficiently.
- Understanding the impact of different control structures on code performance and efficiency.
- Applying control structures in solving practical problems

STRUCTURE OF THE LESSION:

6.1 INTRODUCITON

There are important ways to direct the flow of processing in programmes using R's control structures. There are conditional statements in these frameworks, such as if-else statements, that tell the programme what to do based on certain conditions. Loops, like for and while loops, let you run the same block of code over and over again. This makes it easier to do things like iterate over data items or do calculations over and over again. R also has switch lines for situations where you need to make decisions with more than one branch. Mastering control structures helps programmers write more flexible and effective code, which is important for many tasks, from working with data and analyzing it to solving algorithmic problems. Understanding these structures not only makes code easier to read and manage, but it also lets you make R programmes that are smarter and more flexible

6.2 IF STATEMENT

An expression that is a Boolean and a collection of statements make up this control statement, which is one of the control statements in the R programming language. The collection of statements is carried out in the event that the Boolean expression contains a value of TRUE.

The sentences that come after the conclusion of the If statement are carried out if the Boolean expression is evaluated to show that it is false.

In the following, you will find the fundamental syntax for the If statement::

```
if(expression)
{
# code to execute if condition is TRUE
}
```

Example Program:

```
m <-"Lakshmi Shourya"
if(is.character(m)) {
print("m is a Character")
}
```

6.2.1Else Statement

In the If -Else statement, an If statement is followed by an Else statement, which contains a block of code to be executed when the Boolean expression in the If the statement evaluates to FALSE.'

The basic syntax of it is given below:

```
if(expression)
{
This block of code executes if the Boolean
expression returns TRUE.
}
else
{
This block of code executes if the Boolean
expression returns FALSE.
}
```

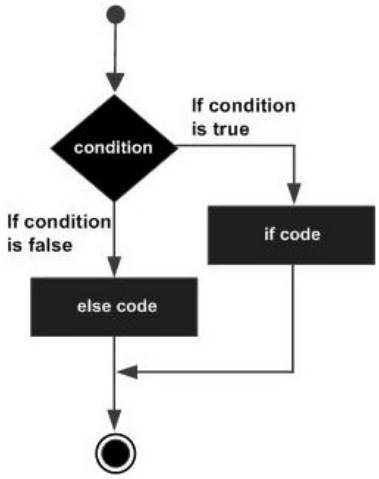


Figure 6.1 if else statement control flow

Example:

```
s <-c("Acharya ","Nagarjuna","University")
if("Nagarjuna"%in%s)
{
      print("Intellipaat")
}
else
{
      print("Not found")
}
```

6.2.2 Else If Statement

An "else if" statement is positioned between a "if" statement and a "else" statement. Successive Else-If statements can be appended to an initial If statement. If an If statement or an Else if statement evaluates to TRUE, none of the remaining else if or Else statements will be processed.

Below is the fundamental syntax of it:

```
if(expression1)
{
        If this block of code executes if the expression 1 returns TRUE
 }
else if(expression2)
{
This block of code executes if the expression 2 returns TRUE
}
else if(expression3)
{   This block of code executes if the expression returns TRUE
}
else
        {
        This block of code executes if none of the expression returns TRUE
        }
```

6.3 THE FOR STATEMENT

A loop is defined as a situation where we need to execute a block of code several number of times. In the case of loops, the statements are executed sequentially.

A for loop in R is used to iterate over a sequence of values or elements and perform a set of operations for each iteration.

The basic syntax of a for loop is given below

```
for (value in vector)
{
        statements
}
```

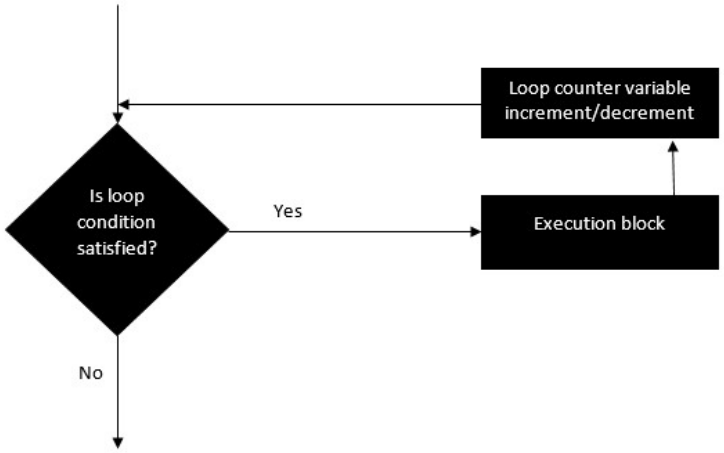


Figure 6.2for loop control flow

Example:

```
v <- c(1:10)
for (i in v)
{
     print(i)
}

Output:
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

We can also use the break statement inside a for-loop to break it out abruptly.

Example:

```
v <- c(1:5)
for (i in v)
{
```

```
     if(i == 3)
    {
          break
   }
print(i)
}

Output:
[1] 1
[1] 2
```

6.4 THE SWITCH STATEMENT

The switch statement is a control statement in R programming that is used to compare a variable with a set of values. Every individual value is referred to as a case.

The basic syntax for a switch statement is as follows:

```
switch(expression, case1, case2, case3....)
```

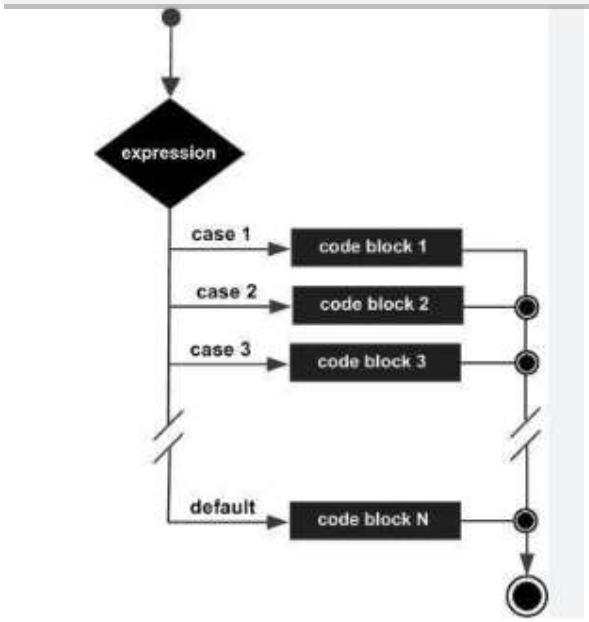


Figure 6.3 switch case statement flow

Example:

```
st <- switch( 3,"Acharya","Nagarjuna","University") )

print(st)

Output:[1] "Nagarjuna"
```

if the value supplied as an expression is not a character string, it is converted to an integer and compared with the indexes of cases specified in the switch statement.

```
m<- "11"

b <- switch(y,"39" = "Srinivas","11"= "Rohit Arya",     "12" =
"Shourya", "40"= "Lakshmi")

print(b)

Output:[1] "Rohit Arya"
```

If an expression evaluates to a character string, it is compared (exactly) to the names of the cases specified in the switch statement.

- If many matches are found, the first matching element is returned.
- There is no default parameter available.

6.5 THE `WHILE` LOOP

A while loop is one of the control statements in R programming which executes a set of statements in a loop until the condition (the Boolean expression) evaluates to TRUE.

The basic syntax of a while loop is given below

```
while(expression)
{
statement
}
```

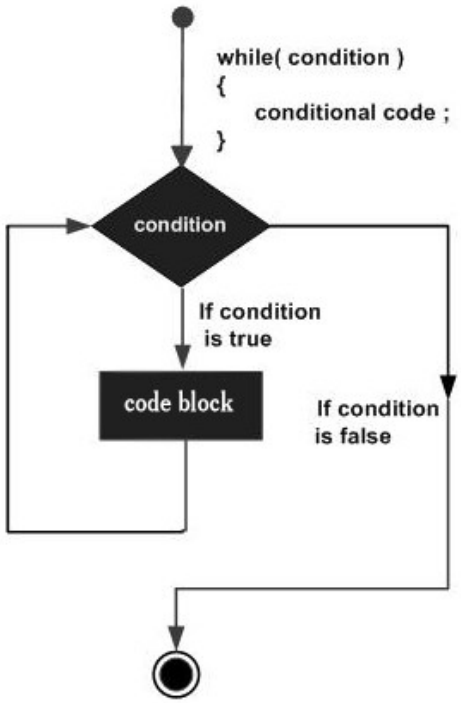


Figure 6.4 while loop control flow

Example:

```
      i <- 1
      while(i <= 10)
      {
      print( i )
        i  = i + 1
      }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

6.6 THE REPEAT AND BREAK STATEMENTS

6.6.1 Repeat

The break statement is used to exit a loop prematurely. It can be used within loops such as for, while, or repeat loops to immediately terminate the loop's execution and continue with the code following the loop. The break statement is typically used in conjunction with a conditional statement to specify the condition under which the loop should be exited.

Example:

```
v <- c(0:6)

for (i in v)
{
     if(i == 3)
    {
         next
    }
    print(i)
}

Output:
[1] 0
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
```

6.6.2 break

The break statement can be used within while or repeat loops to achieve early termination based on certain conditions. It's important to use the break statement judiciously to avoid

unintended behavior, such as infinite loops. Additionally, the break statement only exits the innermost loop it is contained within; if nested loops are present, it will only exit the loop in which it is directly contained.

A break statement is used for two purposes

- To terminate a loop immediately and resume at the next statement following the loop.
- To terminate a case in a switch statement.

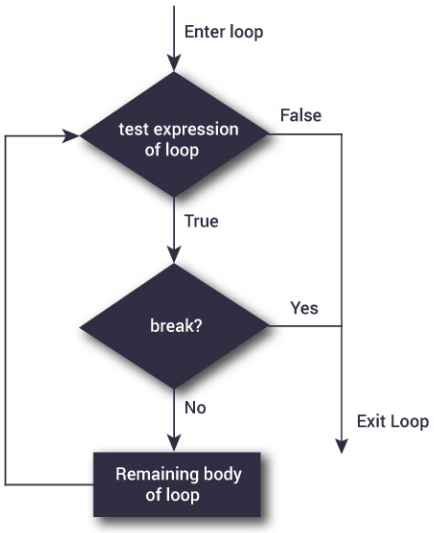


Figure 6.5 flow chart of break statement

```
v <- c(0:6)
for (i in v)
{
if(i == 3)
{
break
}
print(i)
}

Output:
[1] 0
[1] 1
[1] 2
```

6.7 THE NEXT STATEMENT

In R programming, the "next" statement is a control statement that allows for the skipping of the current iteration of a loop, while still continuing the loop. When a "next" statement is reached, the code's further evaluation is eliminated, and the loop proceeds to the next iteration.

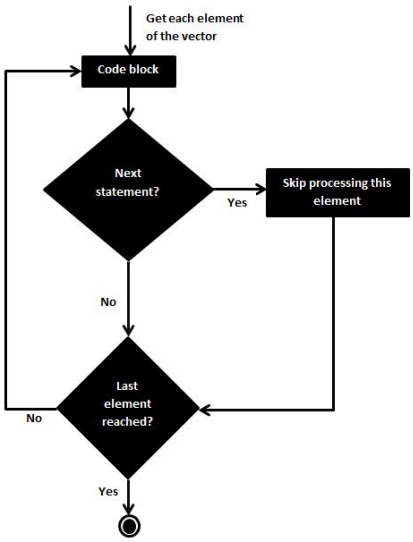


Figure 6.6 next statement flow chart

Example:

```
v <- c(0:6)
for (i in v) {
if(i == 3){
next
}
print(i)
}
Output:

[1] 0
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
```

6.8 SUMMARY

This chapter explain about control structures are fundamental components of R programming, enabling the execution of code in a controlled manner based on specified conditions. Through the use of conditional statements such as if, else, and switch, along with iterative constructs like for, while, and repeat loops, R provides powerful mechanisms for flow control and algorithmic design. These structures allow for the creation of flexible and efficient scripts and functions, facilitating tasks ranging from simple decision-making to complex data processing and manipulation. By mastering R's control structures, programmers gain the ability to write expressive, scalable, and maintainable code, empowering them to tackle diverse analytical challenges across various domains with precision and effectiveness.

1.7 TECHNICAL TERMS

If, for, while, next, break, repeat

1.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What role do logical operators play in conditional statements, and how are they used in conjunction with `if` statements?
2. How can control structures be combined to create complex decision-making and iterative processes in R scripts?
3. Can you compare and contrast control structures in R with those in other programming languages, highlighting similarities and differences?

Short Questions:

1. What are control structures in R, and why are they important in programming?
2. ow do `else` and `else if` statements extend the functionality of `if` statements in R?
3. How does the repeat loop differ from while and for loops, and in what scenarios is it particularly useful?
4. What is the syntax and usage of the while loop in R

1.9SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team
3. Crawley, M. J. (2012). The R book. John Wiley & Sons.
4. Albert, J. & Rizzo, M. (2012). R by Example. Springer
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R. Cambridge.

AUTHOR: **Dr. U. Surya Kameswari**

LESSON- 7

FUNCTIONS

OBJECTIVES:

After going through this lesson, the student will be able to

1 Able to define functions in R
2 Comprehend the importance of functions in organizing code and facilitating code reuse
3 How to handle parameters within functions
4 Apply functions effectively in data analysis tasks
5 Gain proficiency in debugging functions, identifying and resolving common errors

STRUCTURE OF THE LESSION:

7.1 FUNCTIONS

A function in programming is a named block of code that performs a specific task or set of tasks. It encapsulates a sequence of statements and can accept inputs, called arguments or parameters, and optionally return an output. Functions allow programmers to modularize code, making it easier to manage, debug, and reuse. They promote code reusability and abstraction by encapsulating common operations or algorithms into self-contained units.

Functions in R programming are essential components that encapsulate sets of instructions to perform specific tasks or calculations. Whether built-in or user-defined, functions play a crucial role in enabling code modularity, reusability, and abstraction. With R's rich set of built-in functions covering a wide range of operations, from basic arithmetic to advanced

statistical analysis, programmers can leverage these functions to streamline their coding processes and focus on higher-level problem-solving. Moreover, R allows users to define their own functions tailored to their unique requirements, enabling customization and flexibility in analytical workflows. By encapsulating code within functions, R programmers can create modular, maintainable, and efficient codebases that are easier to understand, debug, and extend.

Furthermore, functions in R support a variety of advanced features, including the ability to accept multiple arguments, specify default parameter values, and return multiple outputs. This flexibility allows programmers to create versatile functions that can handle diverse input scenarios and produce customized outputs. Additionally, R functions support lexical scoping, enabling access to variables defined in their enclosing environments. This feature facilitates the creation of more complex and hierarchical functions, enhancing code organization and readability. Overall, functions are a cornerstone of R programming, empowering users to write clear, concise, and reusable code for data analysis, statistical modeling, and beyond.

R has a large number of in-built functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

7.2 FUNCTION DEFINITION

A function definition in programming refers to the process of creating a named block of code that performs a specific task or set of tasks. This block of code encapsulates a sequence of statements and can accept inputs, known as arguments or parameters, to operate on. In the definition, We specify the function's name, the parameters it expects (if any), and the operations it performs. Additionally, We may specify the return type and value of the function, indicating what the function will produce as output when called. Overall, a function definition establishes the blueprint or structure of the function, outlining how it behaves and what it accomplishes when invoked within a program.

An R function is created by using the keyword function. The basic syntax of an R function definition is as follows:

The following is the syntax for a user-defined function in R:

```
Function_name =  function(arguments){
  function_body
  return (return)
}
```

Where **function_name** is the name of the function,

arguments are the input arguments needed by the function,

function_body is the body of the function,

return is the return value of the function.

7.3 FUNCTION COMPONENTS

The different parts of a function are:

function_name: This is the name of the function, which should be descriptive and indicative of its purpose.

arguments: These are the arguments or parameters that the function accepts as input. They represent the values that the function operates on.

Body of the function: This is the sequence of statements or expressions that define the task the function performs. It contains the actual code that executes when the function is called.

return(): This function is used to specify the value or result that the function will return when called. It is optional; if omitted, the function returns the result of the last evaluated expression.

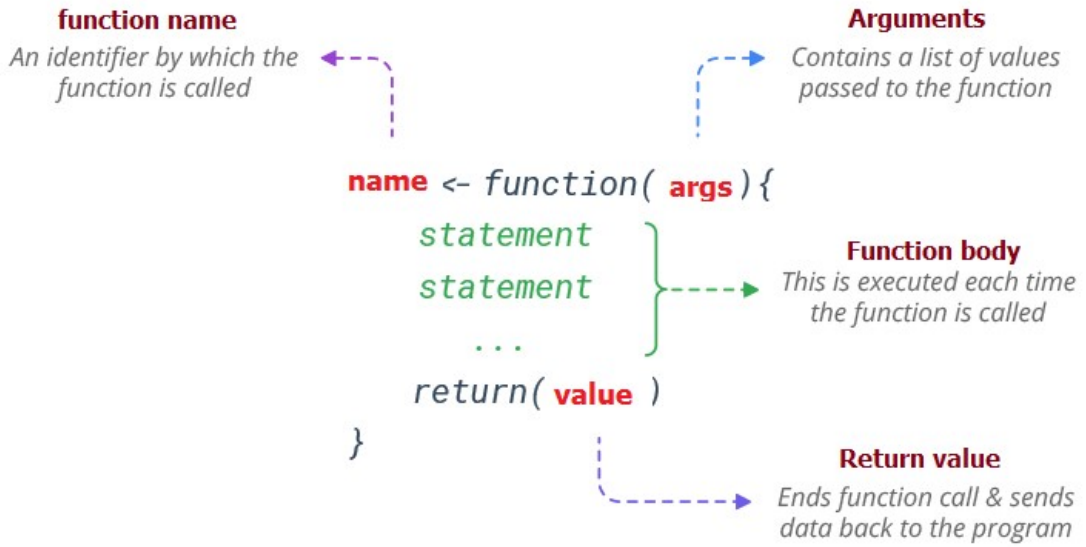


Figure 7.1 components of a function

R has many in-built functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as user defined functions.

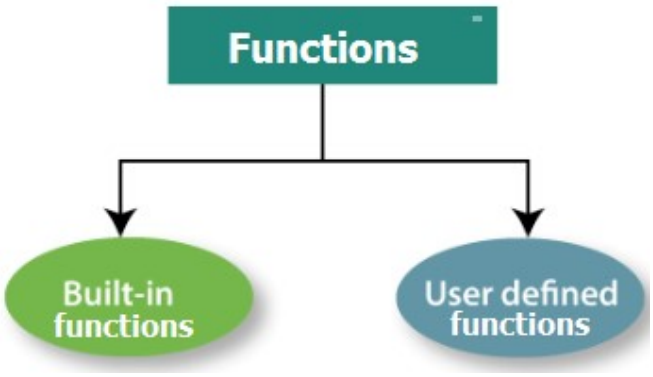


Figure 7.2 types of functions

7.4 BUILT-IN FUNCTIONS

Simple examples of in-built functions are seq(), mean(), max(), sum(x)and paste(...) etc. They are directly called by user written programs.

Built-in functions in R serve several important purposes that contribute to the efficiency and productivity of programming in the language:

Basic Operations: Built-in functions provide essential functionality for performing basic operations such as arithmetic calculations (+, -, *, /), logical operations (&, |, !), and comparisons (==, <, >, <=, >=). These operations form the foundation of data manipulation and analysis in R.

Data Manipulation: R's built-in functions offer powerful tools for manipulating data, such as sorting (sort()), filtering (subset()), merging (merge()), and aggregating (aggregate()). These functions enable users to preprocess and transform datasets efficiently.

Statistical Analysis: R includes a vast array of built-in functions for statistical analysis, including descriptive statistics (mean(), median(), sd()), hypothesis testing (t.test(), wilcox.test()), regression analysis (lm()), and more. These functions facilitate comprehensive data analysis and modeling tasks.

Data Visualization: R's built-in functions support data visualization through packages such as graphics, ggplot2, and lattice. These functions allow users to create a wide range of plots and visualizations to explore and communicate data effectively.

File and Data I/O: R provides built-in functions for reading and writing data to various file formats, including CSV (read.csv(), write.csv()), Excel (read_excel(), write_excel()), and databases (dbReadTable(), dbWriteTable()). These functions simplify data import and export tasks.

Examples:

7.4.1 Numeric Functions

Function	Description
abs(*x*)	absolute value

sqrt(*x***)**	square root
ceiling(*x***)**	ceiling(3.475) is 4
floor(*x***)**	floor(3.475) is 3
trunc(*x***)**	trunc(5.99) is 5
round(*x* **, digits=** *n***)**	round(3.475, digits=2) is 3.48
signif(*x* **, digits=** *n***)**	signif(3.475, digits=2) is 3.5
cos(*x***), sin(***x***), tan(***x***)**	also acos(*x*), cosh(*x*), acosh(*x*), etc.
log(*x***)**	natural logarithm
log10(*x***)**	common logarithm
exp(*x***)**	e^*x*

Character Functions

Function	Description
substr(*x* **, start=** *n1* **, stop=** *n2***)**	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "22222" is "a222ef"
grep(*pattern* **,** *x* **,** **ignore.case=** FALSE **,** **fixed=** FALSE **)**	Search for *pattern* in *x*. If fixed =FALSE then *pattern* is a regular expression. If fixed=TRUE then *pattern* is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2
sub(*pattern* **,** *replacement* **,** *x* **,** **ignore.case** = FALSE **,** **fixed=** FALSE **)**	Find *pattern* in *x* and replace with *replacement* text. If fixed=FALSE then *pattern* is a regular expression. If fixed = T then *pattern* is a text string. sub("\s","."."Hello There") returns "Hello.There"
strsplit(*x* **,** *split***)**	Split the elements of character vector *x* at *split*. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using *sep* string to separate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())
toupper(*x***)**	Uppercase
tolower(*x***)**	Lowercase

Statistical Probability Functions

Function	Description

dnorm(*x***)**	normal density function (by default m=0 sd=1)
	# plot standard normal curve
	x <- pretty(c(-3,3), 30)
	y <- dnorm(x)
	plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i")

pnorm(*q***)**	cumulative normal probability for q (area under the normal curve to the left of q) pnorm(1.96) is 0.975
qnorm(*p***)**	normal quantile. value at the p percentile of normal distribution qnorm(.9) is 1.28 # 90th percentile
rnorm(*n***, m=0, sd=1)**	n random normal deviates with mean m and standard deviation sd. #50 random normal variates with mean=50, sd=10 x <- rnorm(50, m=50, sd=10)

dbinom(*x***,** *size***,** *prob***)**
pbinom(*x***,** *size***,** *prob***)**
qbinom(*p***,** *size***,** *prob***)**
rbinom(*n***,** *size***,** *prob***)**
	binomial distribution where size is the sample size
	and prob is the probability of a heads (pi)
	# prob of 0 to 5 heads of fair coin out of 10 flips
	dbinom(0:5, 10, .5)
	# prob of 5 or less heads of fair coin out of 10 flips
	pbinom(5, 10, .5)

dpois(*x***,***lamda***)**
ppois(*x***,***lamda***)**
qpois(*x***,***lamda***)**
rpois(*x***,***lamda***)**
	poisson distribution with m=std=lamda
	#probability of 0,1, or 2 events with lamda=4
	dpois(0:2, 4)
	# probability of at least 3 events with lamda=4
	1- ppois(2,4)

dunif(*x***, min=0, max=1)** **punif(***x***, min=0, max=1)** **qunif(***x***, min=0, max=1)** **runif(***x***, min=0, max=1)**	uniform distribution, follows the same pattern as the normal distribution above. #10 uniform random variates x <- runif(10)

Other Statistical Functions

Function	Description
mean(*x* **, trim=***0***, na.rm=** FALSE **)**	mean of object x # trimmed mean, removing any missing values and # 5 percent of highest and lowest scores mx <- mean(x,trim=.05,na.rm=TRUE)
sd(*x***)**	standard deviation of object(x). also look at var(x) for variance and mad(x) for median absolute deviation.
median(*x***)**	median
quantile(*x* **,** *probs***)**	quantiles where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0,1]. # 30th and 84th percentiles of x y <- quantile(x, c(.3,.84))
range(*x***)**	range
sum(*x***)**	sum
diff(*x* **, lag=** *1***)**	lagged differences, with lag indicating which lag to use
min(*x***)**	minimum
max(*x***)**	maximum
scale(*x* **, center=** TRUE **, scale=** TRUE **)**	column center or standardize a matrix.

Other Useful Functions

Function	Description
seq(*from* **,** *to* **,** *by***)**	generate a sequence indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9)
rep(*x* **,** *ntimes***)**	repeat *x n* times y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)
cut(*x* **,** *n***)**	divide continuous variable in factor with *n* levels y <- cut(x, 5)

7.5 USER-DEFINED FUNCTION

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

7.5.1 Create a Function

To define a function in R, use the function command and assign the results to a function name.

```
# Create a function 'msgFunc'

msgFunc<- function() {

  print("Acharya Nagarjuna University"")
```

*

If We have only one statement to execute, We can skip curly braces.

```
msgFunc <- function() print('Acharya Nagarjuna University')
```

7.5.2 Call a Function

We can call (run) the function by adding parentheses after the function's name.

```
msgFunc <- function() {
  print('Acharya Nagarjuna University')
}

msgFunc()
[1] "Acharya Nagarjuna University"
```

7.5.3 Pass Arguments

We can send information to a function through arguments. Arguments are declared after the function keyword in parentheses.

We can send as many arguments as we like, just separate them by a comma.

```
sum <- function(x, y)
{
  x + y
}

sum(45, 89)
[1] 134
```

7.5.4 Named Arguments

If we pass arguments to a function by name, we can put those arguments in any order.

```
pow <- function(x, y) {
  x ^ y
}

# using argument names
pow(x=2, y=3)
[1] 8

# changing the order
pow(y=3, x=2)
[1] 8
```

7.5.5 Default Argument Value

We can assign a default value to an argument. So, when We call the function without argument, it uses the default value.

```
# Set default value '3' to second argument
pow <- function(x, y=3)
{
  x ^ y
}
```

function will use default y value

```
pow(2)

[1] 8

# specifying a different y value
pow(2, 4)

[1] 16
```

7.5.6 Return a Value

To return a value from a function, simply use a return() function.

```
sum <- function(x, y) {
  return(x + y)
}

sum(2, 3)
[1] 5
If We do not include any return() function, it automatically returns
the last expression.
sum <- function(x, y) {
  x + y
}
```

```
sum(2, 3)
[1] 5
```

7.5.7 Return Multiple Values

We can return multiple values by saving the results in a vector (or a list) and returning it.

```
math <- function(x, y)
{
  add <- x + y
  sub <- x - y
  mul <- x * y
  div <- x / y
  c(addition = add, subtraction = sub
 multiplication = mul, division = div)
}

math(6, 3)
      addition  subtraction   multiplication    division
             9         3          18            2
```

7.5.8 Lazy Evaluation

+R functions perform lazy evaluation that dramatically extends the expressive power of functions. It is the technique of not evaluating arguments unless and until they are needed in the function.

```
msgFunc <- function(x, y) {
  if(!x){
    return(y)
  }
  else{
    return(x)
  }
}
```

```
# y is not evaluated so not including it causes no harm
msgFunc(6)
[1] 6
```

```
# y is evaluated so not including it raises error
msgFunc(0)
Error in msgFunc(0) : argument "y" is missing, with no default
```

7.5.9 Variable Length Argument

In R, it is often convenient to accept a variable number of arguments passed to the function. To do this, We specify an ellipsis (...) in the arguments when defining a function.

It accepts variable number of arguments, in the sense that We do not know beforehand how many arguments can be passed to Wer function by the user.

For example, below function prints the first argument and then passes all the other arguments to the summary() function.

```
msgFunc <- function(x,...) {print(x); summary(...)}
t<- 1:10

msgFunc("Summary of t:", t)
[1] "Summary of t:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
```

We can also directly refer to the arguments within the argument list (...) through the variables ..1, ..2, to ..9.

For example, ..1 refers to the first argument, ..2 refers to the second, and so on.

```
msgFunc <- function(...) {cat(..1, ..2)}

msgFunc("Acharya", "Nagarjuna University")
Hello World!
```

It is also possible to read the arguments from the argument list by converting the object (...) to a list within the function body.

For example, below function simply sums all its arguments:

```
addAll <- function(x,...) {
  args <- list(...)
  for (a in args) x <- x + a
  x
}

addAll(1,2)
[1] 3

addAll(1,2,3,4,5)
[1] 15
```

7.6 SUMMARY

In the chapter on functions in R, fundamental concepts and techniques for creating and using functions within the R programming language are explored. Functions are essential tools for organizing code, encapsulating specific tasks, and facilitating code reuse. The chapter covers the syntax for defining functions, including parameter specifications and return values, as well as various techniques for enhancing function functionality, such as handling default parameter values, variable scope, and the use of control structures within functions. Overall, understanding functions in R is crucial for developing efficient and maintainable code in data analysis, statistical modeling, and other domains where R is used.

7.7 TECHNICAL TERMS

Functions, Lazy evaluation, arguments

7.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. List out some built-in functions.
2. Explain Lazy evaluation
3. Define Function? List out the function components

Short questions:

1. What is a function in R and why are they important?
2. How do you define a function in R? Provide an example.
3. Compare the types of functions in R programming?

7.9SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team
3. Crawley, M. J. (2012). The R book. John Wiley & Sons.
4. Albert, J. & Rizzo, M. (2012). R by Example. Springer
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R. Cambridge.

AUTHOR: **Dr. U. Surya Kameswari**

LESSON- 8

ARRAYS

OBJECTIVES:

After going through this lesson, you will be able to

- Define the concept of arrays in programming.
- Demonstrate the creation of arrays in R
- Describe the significance of naming columns and rows in arrays.
- Introduce techniques for accessing array elements using named columns and rows
- Introduce methods for performing calculations across array elements

STRUCTURE OF THE LESSION:

8.1 WHAT IS AN ARRAY

Arrays provide a convenient way to store data that is organized in multiple dimensions, such as matrices or higher-dimensional data sets. This can include data from various sources such as experimental measurements, simulation outputs, or observational studies.

Arrays in R are efficient in terms of memory usage and computational performance, especially for large datasets. They allow for fast access to elements and support vectorized operations, which can significantly speed up computations compared to using loops. They are compatible with various data formats commonly used in data analysis and scientific computing, such as CSV files, databases, and spreadsheets. This compatibility makes it easier to import and export data between R and external sources while preserving its multidimensional structure.

Arrays allow for the representation and analysis of data in more than two dimensions, which is essential for certain types of analyses, such as image processing, time-series analysis, and

spatial data analysis. R's array handling capabilities make it well-suited for working with such complex data structures.

They can store data in a structured format, making it easier to explore and visualize multidimensional datasets. R provides powerful tools for data visualization, and arrays can be directly used with these tools to create informative plots and visualizations.

In R programming, an array is a data structure that can hold values of the same type organized in multiple dimensions. Arrays can be thought of as matrices extended to multiple dimensions. They are particularly useful for storing and manipulating multi-dimensional data, such as images, time series, or spatial data. In R, arrays can have two or more dimensions.

The only difference between vectors, matrices, and arrays are

- Vectors are uni-dimensional arrays
- Matrices are two-dimensional arrays
- Arrays can have more than two dimensions

There are two types of arrays in R

- Homogeneous Array: An array where all elements are of the same data type.
- Heterogeneous Array: An array where elements can be of different data types. However, this is less common in R, as R typically enforces homogeneity in arrays.

8.2 ARRAY CREATION

In R, we use the array() function to create an array.

The syntax of the array() function is

```
array(vector, dim = c(nrow, ncol, nmat))
```

Here,
vector - the data items of same type
nrow - number of rows
ncol - number of columns
nmat - the number of matrices of nrow * ncol dimension

Example: Creation of 2 X 3 matrix

```
array1 <- array(c(1:12), dim = c(2,3,2))
print(array1)

Output

, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
```

```
[1,]    7    9   11
[2,]    8   10   12
```

In the above example, we have used the array() function to create an array named array1.

Observe the arguments passed inside array(),

array(c(1:15), dim = c(2,3,2))
where,
c(1:12) - a vector with values from 1 to 12
dim = c(2,3,2) - create two matrices of 2 by 3 dimension

Finally, the numbers from 1 to 12 that are arranged in two 2 by 3 matrices are printed.

8.3 NAMING COLUMNS AND ROWS

In R programming, naming columns and rows of arrays (or matrices) can be helpful for several reasons:

Giving meaningful names to columns and rows can make your code more readable and understandable. Named columns and rows serve as a form of self-documentation. By assigning names that describe the data they contain, we provide context and clarity to the structure of your array. This can reduce the need for additional comments in our code, as the names themselves convey important information about the data.

Named rows and columns simplify the process of subsetting or accessing specific elements of the array. Instead of remembering or looking up the numeric indices, we can use the names directly, which can be more intuitive and less error-prone. If there's an issue with your code, having named columns and rows can aid in debugging. Error messages or warnings may refer to the named entities, making it easier to pinpoint the problem areas in your code.

Many R functions support named arguments, allowing you to pass data using column or row names instead of numeric indices. When your array has named columns and rows, you can directly use these names when working with such functions, which can streamline your code and improve its readability.

In R programming, we can name the columns and rows of arrays using the `dimnames()` function. Arrays in R can have two types of names: row names and column names.

Here's how we can name the rows and columns of an array:

```
# Create a 3x3 array
my_array <- array(1:9, dim = c(3, 3))

# Define row and column names
```

```
row_names <- c("Row1", "Row2", "Row3")
col_names <- c("Col1", "Col2", "Col3")


# Assign names to rows and columns

dimnames(my_array) <- list(row_names, col_names)


# Print the array with row and column names
print(my_array)
```

Output:

```
     Col1 Col2 Col3
Row1    1    4    7
Row2    2    5    8
Row3    3    6    9
```

In this example, `my_array` is a 3x3 array with row names "Row1", "Row2", and "Row3", and column names "Col1", "Col2", and "Col3". The `dimnames()` function is used to set these names for the rows and columns of the array.

8.4 ACCESSING ARRAY ELEMENTS

In R, there are several ways to access elements of an array, depending on the dimensionality of the array and the specific elements you want to retrieve. Here are the various methods for accessing array elements:

8.4.1 Using Numeric Indices: we can access array elements using numeric indices, similar to accessing elements in vectors. For a one-dimensional array, you use a single index. For higher-dimensional arrays, you specify indices for each dimension. For example:

```
# One-dimensional array

a <- c(1, 2, 3, 4)
a[3]  # Accessing the third element


# Two-dimensional array

b <- array(1:12, dim = c(3, 4))
b[2, 3]  # Accessing the element in the second row and third column
```

8.4.2. Using Row and Column Names: If our array has named rows and columns, we can access elements using these names. This is particularly useful for improving code readability and making your code more self-explanatory.

For example:

```
# Creating a named two-dimensional array
c <- array(1:12, dim = c(3, 4), dimnames = list(c("row1", "row2", "row3"), c("col1", "col2", "col3", "col4")))
c["row2", "col3"]   # Accessing the element in the second row and third column
```

8.4.3. Using Logical Indexing: You can use logical indexing to select elements based on certain conditions. This is useful for filtering elements that meet specific criteria.

For example:

```
# Creating an array
d <- array(1:12, dim = c(3, 4))

# Selecting elements greater than 5
d[d > 5]
```

8.4.4 . Using Partial Matching: In some cases, R allows you to use partial matching when accessing array elements using names. This can be done by specifying the `exact` parameter as `FALSE`. However, it's generally recommended to avoid partial matching to prevent potential errors and ambiguities. For example:

```
# Accessing an element using partial matching
c["ro", "co", exact = FALSE]  # Accessing the element in the row that starts with "ro" and column that starts with "co"
```

These are some of the common methods for accessing elements of an array in R. Each method has its advantages and use cases, so choose the one that best fits your specific requirements and preferences.

8.5 MANIPULATING ARRAY ELEMENTS

In R, there are several ways to manipulate array elements, including accessing, modifying, and performing operations on them.

The following are some common methods:

8.5.1. Accessing Elements:

We can access individual elements or subsets of elements from an array using indexing. R uses square brackets `[ ]` for indexing.

```
# Create an array
  my_array <- array(1:12, dim = c(3, 4))
# Access individual element
  print(my_array[2, 3])   # Accesses element at row 2, column 3

# Access entire row or column
  print(my_array[2, ])    # Accesses entire second row
  print(my_array[, 3])  # Accesses entire third column
```

2. Modifying Elements:

We can modify elements of an array by assigning new values to them using indexing.

```
# Modify an individual element
  my_array[2, 3] <- 99    # Modifies element at row 2, column 3 to 99

# Modify entire row or column
  my_array[2, ] <- c(10, 20, 30, 40)     # Modifies entire second row
  my_array[, 3] <- 0        # Modifies entire third column to zeros
```

3. Conditional Manipulation:

We can conditionally modify elements based on certain criteria using logical indexing.

```
# Set negative values to zero
  my_array[my_array < 0] <- 0

# Double values greater than 5
  my_array[my_array > 5] <- my_array[my_array > 5] * 2
```

8.5.4. Applying Functions:

We can apply functions to elements of an array using functions like `apply()`, `lapply()`, `sapply()`, etc.

```
# Apply sum function to rows
  row_sums <- apply(my_array, 1, sum)     # Calculates row sums

# Apply mean function to columns
  col_means <- apply(my_array, 2, mean) # Calculates column means
```

8.5.5. Reshaping Arrays:

We can reshape arrays using functions like `aperm()`, `array_reshape()` from packages like `reshape2`, or functions from the `tidyverse` ecosystem.

```
# Reshape array from 3D to 2D

reshaped_array <- array_reshape(my_array, c(3, 4))
```

6. Concatenating and Combining Arrays:

You can concatenate arrays along different dimensions or combine them using functions like `cbind()`, `rbind()`, `abind()` from the `abind` package, or functions from the `tidyverse` ecosystem.

Concatenate arrays along rows

```
combined_array <- rbind(my_array, another_array)


# Concatenate arrays along columns

combined_array <- cbind(my_array, another_array)
```

These are some common methods for manipulating array elements in R. Depending on the specific task and complexity of the operation, you may choose one or more of these methods to effectively manipulate array data in R.

8.6 CALCULATION ACROSS ARRAY ELEMENTS

In R, there are several ways to perform calculations across array elements, leveraging R's vectorized operations and functions designed for array manipulation. Here are some common methods along with examples:

8.6.1. Using Arithmetic Operators:

R's arithmetic operators (`+`, `-`, `*`, `/`, `^`, etc.) can be applied directly to arrays, performing element-wise calculations.

```
# Create two arrays

arr1 <- array(1:9, dim = c(3, 3))
arr2 <- array(9:1, dim = c(3, 3))


# Element-wise addition

result_add <- arr1 + arr2
```

```
# Element-wise multiplication
result_mult <- arr1 * arr2

print(result_add)
print(result_mult)
```

8.6.2. Using Built-in Functions:

R provides a variety of built-in functions that operate on arrays, performing calculations across their elements. Examples include `sum()`, `mean()`, `sd()`, `max()`, `min()`, etc.

```
# Calculate sum across rows (apply function along rows)
row_sums <- apply(arr1, 1, sum)


# Calculate mean across columns (apply function along columns)
col_means <- apply(arr2, 2, mean)


print(row_sums)
print(col_means)
```

8.6.3 Using Matrix Multiplication:

Matrix multiplication can be used to perform calculations across arrays. This is particularly useful for linear algebraic operations.

```
# Create matrices
mat1 <- matrix(1:6, nrow = 2, byrow = TRUE)
mat2 <- matrix(6:1, nrow = 2, byrow = TRUE)
# Perform matrix multiplication
result_matmul <- mat1 %*% mat2
print(result_matmul)
```

4. Using 'sapply()' or 'lapply()':

These functions can be used to apply a function to each element of an array. They iterate over each element of the array, perform the specified operation, and return the result.

```
# Create an array
arr <- array(1:9, dim = c(3, 3))

# Square each element using sapply
result_square <- sapply(arr, function(x) x^2)

# Take square root of each element using lapply
result_sqrt <- lapply(arr, sqrt)
```

```
print(result_square)
print(result_sqrt)
```

8.6.5. Using `rowSums()` and `colSums()`:

These functions calculate row-wise and column-wise sums, respectively, across the elements of an array.

```
# Calculate row-wise sums
row_sums <- rowSums(arr)

# Calculate column-wise sums
col_sums <- colSums(arr)

print(row_sums)
print(col_sums)
```

These methods provide flexible ways to perform calculations across array elements in R, allowing for efficient and concise data manipulation and analysis. Depending on the specific task and requirements, you can choose the most appropriate method for your needs.

8.7 SUMMARY

The chapter delves into the fundamental concepts surrounding arrays in R programming, elucidating their significance and various operations associated with them. It begins by elucidating the essence of arrays, elucidating how they serve as indispensable data structures for storing collections of elements, which can be accessed via indices or keys. Array creation is then explored, shedding light on techniques for generating arrays and assigning names to their rows and columns, enhancing readability and facilitating subsequent operations. The chapter progresses to elucidate methods for accessing and manipulating array elements, elucidating diverse strategies for efficient data retrieval and modification. Furthermore, it delves into the intricate realm of conducting calculations across array elements, delineating diverse approaches such as arithmetic operators, built-in functions, matrix multiplication, and iterative processes, which empower users to perform complex computations with ease. Through elucidating these pivotal topics, the chapter equips readers with a comprehensive understanding of arrays in R programming and instills proficiency in array manipulation for diverse data analysis endeavors.

8.8 TECHNICAL TERMS

Array, R programming

8.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What is the definition of an array in programming?
2. Describe the process of creating an array in R.
3. How do you access a specific element in a multidimensional array?

Short Questions:

1. How can you perform calculations across array elements in R?
2. Provide examples of using functions like apply(), sapply(), and lapply() for array manipulation.
3. How can you perform matrix multiplication in R?
4. Describe the process of manipulating array elements in R.

8.10 SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. Crawley, M. J. (2012). The R book. John Wiley & Sons.
3. Albert, J. & Rizzo, M. (2012). R by Example. Springer
4. Teetor, P. (2011). R Cookbook. O'REILLY
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R.Cambridge.

AUTHOR: **Mr. G V Suresh**

LESSON- 9
VECTORS

OBJECTIVES:

After going through this lesson, you will be able to

- Comprehend the concept of vectors as one-dimensional arrays for data storage and manipulation in R
- Create vectors using various methods
- Categorize vectors into different types based on their data content
- Learn to access specific elements within vectors using indexing techniques
- Gain proficiency in utilizing a range of built-in functions that operate on vectors

STRUCTURE OF THE LESSION:

9.1 INTRODUCTION

In R programming, vectors serve as fundamental building blocks for data manipulation and analysis. A vector in R is a one-dimensional array that can hold elements of the same data type, such as numeric, character, or logical values. Understanding vectors is crucial because many operations in R are vectorized, meaning they are designed to work efficiently with entire vectors rather than individual elements. This vectorized approach enhances the speed and simplicity of data processing tasks, making R a powerful tool for statistical computing and data analysis.

Furthermore, vectors in R support vectorized operations, meaning arithmetic operations and functions can be applied element-wise to entire vectors. This capability simplifies code and improves computational efficiency, as it eliminates the need for explicit looping constructs. For example, adding two vectors together with the `+` operator results in element-wise addition, making it straightforward to perform calculations on entire datasets at once. Understanding how to leverage vectorized operations is essential for writing efficient and

concise code in R, enabling analysts and programmers to streamline their workflows and focus on extracting insights from data.

The Vectors in R is the simplest basic type of object that is used to store a sequence of data elements of the same type. Members of a Vector are called Components

9.1.1 Features

- Vector is a basic data structure in R.
- It is a one-dimensional data structure.
- It holds elements of the same type.
- Members in the vector are called components.
- It is not recursive.
- We can easily convert vectors into data frames or matrices.
- In DataFrame, each column is considered a vector.

56	43	78	29	30	46

9.1.2 Applications:

- The usage of vectors is common in machine learning for the purpose of principal component analysis. After that, they are utilized for the purpose of carrying out decomposition in vector spaces after being extended to include eigenvalues and eigenvectors.

- It is in the form of vectors that the inputs that are provided to the deep learning model are.
- The neural network's input layer receives these vectors, which are made up of standardized data. These vectors are delivered to the neural network.

- Vectors are utilized in the process of working on the construction of support vector machine algorithms.

- A wide variety of tasks, including image recognition and text processing, are carried out by neural networks through the utilization of vector operations.

9.2 CREATION OF VECTOR

In R, you can create a vector using the `c()` function, which stands for "combine" or "concatenate." This function allows you to combine individual elements into a vector.

This function returns a one-dimensional array or simply vector. The c() function is a generic function which combines its argument. All arguments are restricted with a common data type which is the type of the returned value

```
# create vector of string types

employees <- c("Surya", "Shourya", "Arya")
```

```
print(employees)


# Output: [1] "Surya" "Shourya" "Arya"
```

In the above example, we have created a vector named employees with elements: Surya , Shourya and Arya. Here, the `c()` function creates a vector by combining three different elements of employees together.

There are various other ways to create a vector in R, which are as follows:

9.2.1: Using the colon(:) operator

We can create a vector with the help of the colon operator. There is the following syntax to use colon operator:

```
z<-x:y
```

This operator creates a vector with elements from x to y and assigns it to z.

Example:

```
a<-4:-10
a
Output
[1]   4   3   2   1   0   -1   -2   -3   -4   -5   -6   -7   -8   -9
-10
```

9.2.2 Using the seq() function

In R, we can create a vector with the help of the seq() function. A sequence function creates a sequence of elements as a vector. The seq() function is used in two ways, i.e., by setting step size with `'by'` parameter or specifying the length of the vector with the `'length.out'` feature.

Example:

```
seq_vec<-seq(1,4,by=0.5)
seq_vec
class(seq_vec)


Output
[1]   1.0   1.5   2.0   2.5   3.0   3.5   4.0
```

Example:

```
seq_vec<-seq(1,4,length.out=6)
seq_vec
```

```
class(seq_vec)


Output
[1]   1.0   1.6   2.2   2.8   3.4   4.0
[1]   "numeric"
```

9.2.3 Creating a vector with repeated elements

The rep() method of base R is used to generate a replicated sequence from a specified vector, where each element of the vector can be repeated at any number of specified times. This method can take a character, floats, or integers-type input vectors.

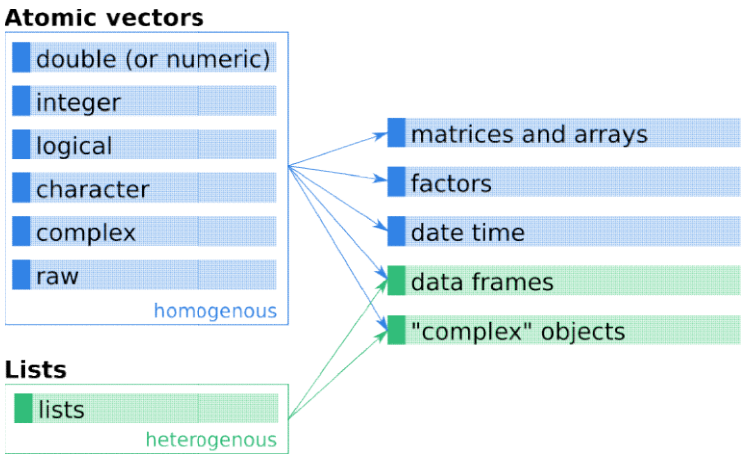
Example:

```
# Repeat vectors
v_rep = rep(c(2, 8), times = 5)
print(v_rep)

# Output:
#[1] 2 8 2 8 2 8 2 8 2 8
```

9.3 TYPES OF VECTORS

In R, there are several types of vectors, each with its own characteristics and uses. The main types of vectors are:



Source: discdown.org

9.3.1 Numeric Vectors: Numeric vectors are used to store numerical data, such as integers or decimal numbers. Numeric vectors are created using the 'c()' function or by specifying a sequence of numbers using the ':' operator. For example:

Example:

```
vec1<- c(24, 57, 82, 90)
# display type of vector
typeof(vec1)

Output:
[1] "double"
[1] "integer"
```

2. Character Vectors: Character vectors are used to store text data, such as strings of characters. Character vectors are created using the 'c()' function with elements enclosed in double or single quotes. In character vectors, by default numeric values are converted into characters.

Example:

```
v1<- c('surya kameswari',  'assistant professor', 'CSE', '20091')

# displaying type of vector

typeof(v1)

Output:

[1] "character"
```

3. Logical Vectors: Logical vectors are used to store boolean values ('TRUE' or 'FALSE'). They are often generated as a result of logical operations or comparisons. Logical vectors are created using the 'c()' function or logical operations

Example:

```
v1<- c(TRUE, FALSE, TRUE, NA)

# Displaying type of vector
typeof(v1)

Output:
[1] "logical"
```

4. Integer Vectors: Integer vectors are used to store integer values. They are a subtype of numeric vectors but differ in that they can only store whole numbers without any decimal places. Integer vectors are created by specifying the 'L' suffix after the number or by coercing numeric vectors using the 'as.integer()' function.

Example:

```
Vec1<- c(24L, 57L, 82L, 90L)
# display type of vector
typeof(vec1)
```

```
Output:
 [1] "integer"
```

9.4 ACCESSING ELEMENTS OF VECTOR

In R, you can access elements in a vector using indexing. Indexing allows you to specify which elements of the vector you want to retrieve or manipulate. Here's how you can access elements in a vector:

1. Single Element Access: To access a single element of a vector, you use square brackets `[ ]` with the index of the element you want to retrieve. R uses 1-based indexing, meaning the first element of a vector has an index of 1. For example:

```
my_vector <- c(10, 20, 30, 40, 50)
first_element <- my_vector[1]  # Accessing the first element
second_element <- my_vector[2]  # Accessing the second element

Output:
[1] 10
[2] 20
```

2. Multiple Elements Access: You can also access multiple elements of a vector by providing a vector of indices inside the square brackets. For example:

Accessing the first, third, and fifth elements

```
my_vector <- c(10, 20, 30, 40, 50)
selected_elements <- my_vector[c(1, 3, 5)]
selected_elements

output:
[1] 10 30 50
```

3. Logical Indexing: You can use logical vectors to subset a vector based on certain conditions. For example, you can create a logical vector indicating which elements meet a specific condition and use that to subset the original vector. For instance:

```
# Creating a logical vector indicating elements greater than 20

my_vector <- c(10, 20, 30, 40, 50)

logical_vector <- my_vector > 20
logical_vector


# accessing elements greater than 20
```

```
elements_greater_than_20 <- my_vector[logical_vector]
elements_greater_than_20
```

Output:

```
[1] FALSE FALSE  TRUE  TRUE  TRUE
[1] 30 40 50
```

4. Negative Indexing: We can use negative indices to exclude certain elements from the vector. For example:

```
my_vector <- c(10, 20, 30, 40, 50)

vector_except_first <- my_vector[-1]  # Excluding the first element
vector_except_first
vector_except_last <- my_vector[-length(my_vector)]  # Excluding the
last element
vector_except_last

Output:
[1] 20 30 40 50
[1] 10 20 30 40
```

9.5 VECTOR OPERATIONS

In R, vectors support a wide range of operations, including arithmetic operations, logical operations, and various functions. Here are some common operations performed on vectors in R:

9.5.1 Combining vectors

The c() function is not only used to create a vector, but also it is also used to combine two vectors. By combining one or more vectors, it forms a new vector which contains all the elements of each vector. Let see an example to see how c() function combines the vectors.

Example:

```
p <- c(15,21,48,53,79,83)
q <- c("Lakshmi", "Shourya" , "Rohit", "Arya", "Sri", "Nivas")
r <- c(p,q)

Output


[1] "15"        "21"        "48"        "53"        "79"        "83"
[7] "Lakshmi", "Shourya" , "Rohit", "Arya", "Sri", "Nivas"
```

9.5.2 Arithmetic operations

We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors. We can add, subtract, multiply, or divide two vectors. Let see an example to understand how arithmetic operations are performed on vectors.

Example:

```
a<-c(1,3,5,7)
b<-c(2,4,6,8)
a+b
a-b
a/b
a%%b

Output
[1]  3  7 11 15
[1] -1 -1 -1 -1
[1]  2 12 30 56
[1] 0.5000000 0.7500000 0.8333333 0.8750000
[1] 1 3 5 7
```

9.5.3 Length of R vector

In R, the length of a vector is determined by the number of elements it contains. we can use the length() function to retrieve the length of a vector.

```
# Create a numeric vector
x <- c(1, 2, 3, 4, 5)


# Find the length of the vector
length(x)
# Create a character vector
y <- c("apple", "banana", "cherry")

# Find the length of the vector
length(y)
# Create a logical vector
z <- c(TRUE, FALSE, TRUE, TRUE)

# Find the length of the vector
length(z)
```

Output:

```
> length(x)
[1] 5

> length(y)
[1] 3

> length(z)
[1] 4
```

9.5.4 Modifying a R vector

Modification of a Vector is the process of applying some operation on an individual element of a vector to change its value in the vector. There are different ways through which we can modify a vector:

```
# Creating a vector
X<- c(32, 87, 90, 87, 98, 32)
X1 <- c(65,80,34,27,18)
# modify a specific element
X[3] <- 21
X[2] <- 90
cat('subscript operator', X, '\n')

# Modify using different logics.
X[1:5]<- 0
cat('Logical indexing', X, '\n')

# Modify by specifying  the position or elements.
X<- X1[c(3, 2, 1)]
cat('combine() function', X)
```

Output:

```
subscript operator 32 90 21 87 98 32
Logical indexing 0 0 0 0 0 32
combine() function 34 80 65
```

Deleting a R vector

Deletion of a Vector is the process of deleting all of the elements of the vector. This can be done by assigning it to a NULL value.

```
# Creating a Vector
M<- c(8, 10, 2, 5)

# set NULL to the vector
M<- NULL
cat('Output vector', M)

Output:
Output vector NULL
```

9.6 FUNCTIONS OPERATED ON VECTORS

In R, a multitude of functions are designed to operate on vectors, allowing for efficient data manipulation, transformation, and analysis. The following are some commonly used functions operated on vectors:

9.6.1 Statistics Functions: These functions provide summary statistics of the elements within a vector:

- mean(): Computes the arithmetic mean of the elements.
- median(): Calculates the median value of the elements.
- sum(): Calculates the sum of the elements.
- min(): Finds the minimum value in the vector.
- max(): Finds the maximum value in the vector.
- range(): Computes the range (minimum and maximum values) of the vector.
- sd(): Computes the standard deviation of the elements.
- var(): Computes the variance of the elements.

```
x <- c(3.16, 4.27, 9.21, 1.35, 8.04)
max(x)
min(x)
mean(x)
median(x)
sum(x)
range(x)
sd(x)
var(x)

Output:
[1] 9.21
[1] 1.35
[1] 5.206
[1] 4.27
```

```
[1] 26.03
[1] 1.35 9.21
[1] 3.316418
[1] 10.99863
```

9.6.2 Mathematical Functions: These functions perform mathematical operations on the elements of a vector:

- sqrt(): Computes the square root of each element.

- log(): Computes the natural logarithm of each element.

- exp(): Computes the exponential of each element.

- abs(): Computes the absolute value of each element.

- round(): Rounds each element to the nearest integer or specified number of decimal places.

Examples:

```
x <- c(3.16, 4.27, 9.21, 1.35, 8.04)
sort(x)
log(x)
exp(x)
abs(x)
round(x)

Output:
[1] 1.35 3.16 4.27 8.04 9.21
[1] 1.1505720 1.4516138 2.2202899 0.3001046 2.0844291
[1]   23.570596   71.521636 9996.596859    3.857426 3102.613190
[1] 3.16 4.27 9.21 1.35 8.04
[1] 3 4 9 1 8
```

9.6.3 Logical Functions: These functions perform logical operations on vectors, returning logical values or indices:

- which(): Returns the indices of the elements that are TRUE.

- all() and any(): Check if all or any elements in a logical vector are TRUE.

- which.min() and which.max(): Returns the index of the minimum or maximum value in a vector.

```
x <- c(23,56,76,32,78,45,39)

#check if all values are less than 10
```

```
all(data < 50)
```

Output:

```
[1] FALSE
```

```
#check if any values are less than 10

any(data < 50)
```

Output:

```
[1] TRUE

which.max(x)
which.min(x)
```

Output:

```
[1] 5
[1] 1
```

9.6.4. Data Transformation Functions: These functions manipulate the structure or content of vectors:

- order(): Returns the indices that would sort a vector.

- unique(): Returns the unique elements of a vector.

- rev(): Reverses the order of elements in a vector.

x <- c(23,56,76,32,78,45,39)

```
x_rev <- rev(x)
x_rev
```

Output:

```
[1] 39 45 78 32 76 56 23
```

order() function ordered the elements of the vector x according to its index

```
order(x)
```

Output:

```
[1] 1 4 7 6 2 3 5
```

In this case, the output would be: 1 4 7 6 2 3 5, which indicates that the smallest value is at index 1, the next smallest is at index 4, and so on and also observe that that the original vector "x" is not modified by this function.

```
#An input vector having duplicate values
y<-c(45,40,3,40,38,6,45,6,84,91,84,6)

#elimnates the duplicate values in the vector
unique(y)

Output:
[1] 45 40 3 38 6 84 91
```

9.7 SUMMARY

The chapter on vectors in R provides a comprehensive overview of this fundamental data structure and its manipulation in the R programming language. It begins with an introduction to vectors, emphasizing their importance as one-dimensional arrays for storing data efficiently. The chapter covers the creation of vectors using various methods such as the `c()` function and sequence generation. Different types of vectors, including numeric, character, logical, integer, and complex, are discussed, highlighting their distinct characteristics and uses. Additionally, the chapter explains how to access elements within vectors using indexing techniques and explores a wide range of operations and functions that can be applied to vectors, including arithmetic and logical operations, statistical functions for data analysis, and transformation functions for data manipulation. Overall, this chapter serves as a comprehensive guide for understanding and working with vectors in R, essential for data manipulation and analysis tasks.

9.8 TECHNICAL TERMS

Vector, Operations, Functions

9.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What is a vector in the context of R programming, and why is it important?
2. How can you create a vector in R? Provide examples using the `c()` function and sequence generation.
3. What are the different types of vectors available in R, and what are their characteristics?
4. How do you access elements within a vector in R? Explain indexing techniques with examples.
5. What operations can be performed on vectors in R? Provide examples of arithmetic, logical, and statistical operations.

Short Questions:

1. What are some commonly used functions operated on vectors in R? Give examples of descriptive statistics, mathematical functions, and probability distributions.
2. How can vectors be manipulated and transformed using functions in R? Provide examples of sorting, unique values, and reversing order.
3. How does R handle vectorization, and why is it important for efficient data processing?
4. Explain the concept of logical indexing in R. How can it be used to subset vectors based on certain conditions?
5. What are some practical applications of vectors and vector operations in real-world data analysis scenarios?

9.10 SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. Crawley, M. J. (2012). The R book. John Wiley & Sons.
3. Albert, J. & Rizzo, M. (2012). R by Example. Springer
4. Teetor, P. (2011). R Cookbook. O'REILLY
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R.Cambridge.

AUTHOR: **Mr. G V Suresh**

LESSON- 10
PACKAGES

OBJECTIVES:

After going through this lesson, you will be able to

- Grasp the concept of R packages as collections of functions, data, and documentation
- Gain knowledge about R repositories, including CRAN, Bioconductor and GitHub.
- Learn various methods for installing R packages
- Differentiate between the terms "package" and "library" within the context of R programming

STRUCTURE OF THE LESSION:

10.1 WHAT IS AN R PACKAGE

An R package is a collection of R functions, data sets, and other resources bundled together for a specific purpose or task. These packages are created by developers to extend the functionality of R and make it easier for users to perform various tasks such as data manipulation, statistical analysis, machine learning, visualization, and more.

R packages typically consist of:

- Functions: R functions that perform specific tasks or computations.
- Documentation: Documentation describing the package and its functions, often including examples and usage guidelines.
- Data sets: Pre-loaded datasets that users can use to test functions or for demonstration purposes.
- Vignettes: Comprehensive documents providing detailed examples and explanations of package usage.
- Dependencies: Packages on which the package relies for functionality, often listed in the DESCRIPTION file.

Packages play a crucial role in the R ecosystem as they allow users to easily access and use specialized tools and functions without having to write code from scratch. Users can install packages from various repositories such as the Comprehensive R Archive Network (CRAN), GitHub, or Bioconductor, and then load them into their R sessions to access the functions and resources provided by the package. Additionally, users can also create their own packages to share their code and functionality with others.

The basic information about a package is provided in the DESCRIPTION file, where you can find out what the package does, who the author is, what version the documentation belongs to, the date, the type of license its use, and the package dependencies.

For example, for the "stats" package, these ways will be:

```
packageDescription("stats")

help(package = "stats")
```

10.2 WHAT ARE R REPOSITORIES?

A repository is a designated location where packages are stored, allowing users to easily install them. In the context of R programming, repositories refer to online platforms or servers where R packages are stored and made available for installation and distribution. These repositories serve as centralized hubs for discovering, downloading, and managing R packages.The most commonly used R repositories include:

These repositories collectively form the backbone of the R ecosystem, providing users with access to a vast array of packages for various purposes, from data analysis and visualization to machine learning and bioinformatics.

10.2.1 CRAN:

It is the official repository, is a network of FTP and web servers that is managed by the global R community. The R foundation oversees the coordination of package publication on CRAN. To be accepted for publication, a package must successfully pass multiple tests to guarantee compliance with CRAN regulations.

10.2.2 Bioconductor

Bioconductor is a specialised repository designed for open-source software in the field of bioinformatics. CRAN operates with its own distinct submission and review procedures, and its community is highly engaged, organising many conferences and meetings year.

10.2.3 Github

Github, while not exclusive to the R programming language, is widely regarded as the most widely used platform for hosting open-source projects. The appeal of this software stems from its ability to provide infinite space for open source projects, seamless interaction with git, a version control software, and its user-friendly features for sharing and collaborating with others. However, it is important to note that there is no formal evaluation mechanism linked to it.

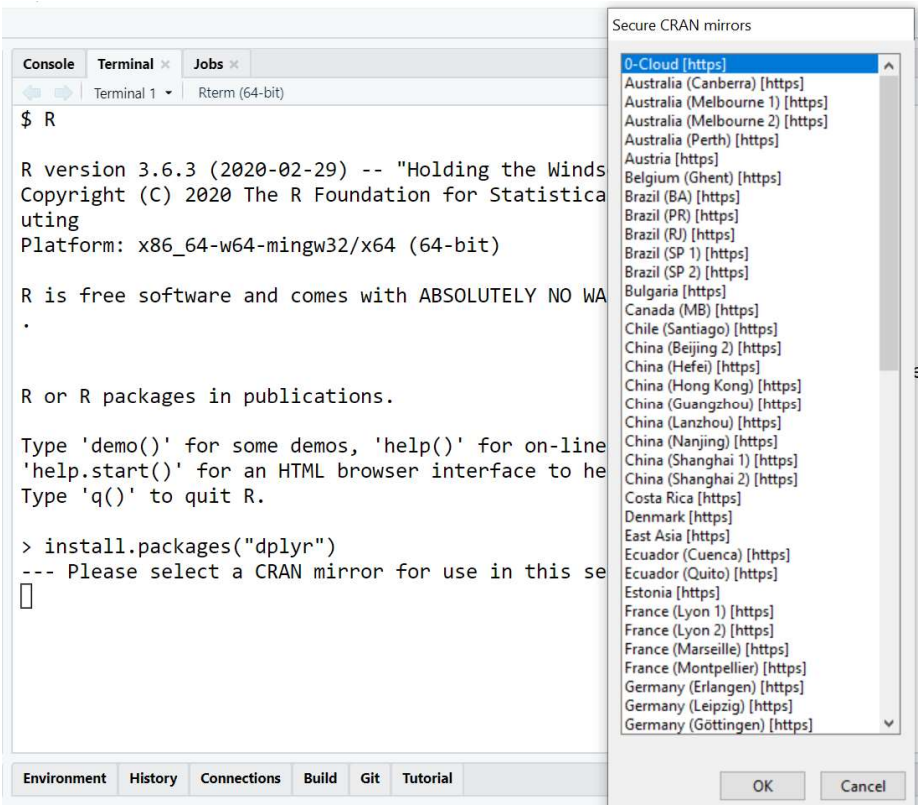
10.3 How to install R packages

10.3.1 Installing R Packages from the CRAN Repository

The Comprehensive R Archive Network (CRAN) repository stores thousands of stable R packages designed for a variety of data-related tasks. Most often, we will use this repository to install various R packages.

To install an R package from CRAN, we can use the install.packages() function:

```
install.packages('readr')
```



Here, we have installed the readr R package used for reading data from the files of different types: comma-separated values (CSV), tab-separated values (TSV), fixed-width files, etc. Make sure that the name of the package is in quotation marks.

We can use the same function to install several R packages at once. In this case, we need to apply first the c() function to create a character vector containing all the desired packages as its items:

```
install.packages(c('readr', 'ggplot2', 'tidyr'))
```

`Install from:` Above, we've installed three R packages: the already-familiar readr, ggplot2 (for data visualization), and tidyr (for data cleaning).

Alternative Method

If we work with R in an IDE, we can use the menu instead of the install.packages() function to install the necessary modules from the CRAN repository. For example, in RStudio, the most popular IDE for R, we need to complete the following steps:

Click `Tools → Install Packages`
Select `Repository (CRAN)` in the slot
Type the package name (or several package names, separated with a white space or comma)
Leave `Install dependencies` ticked as by default
Click `Install`

In the other IDEs for working with R, the buttons and commands will be called differently, but the logic behind them is the same, and all the steps are usually intuitive.

10.3.2 Installing R packages from GitHub

To install an R package from GitHub, e should perform the following steps:

- Install RTools if it hasn't been already installed (otherwise, skip this step)
  - Open https://cran.r-project.org/
  - Select the necessary operating system for downloading the installer (e.g., Download R for Windows)
  - Select `RTools`
  - Select the latest version of RTools
  - Wait for the completion of downloading
  - Run the installer with all the options by default (here we may need to click `Run anyway` on the first pop-up window)
  - For the new versions of R (v4.0.0), add `PATH='${RTOOLS40_HOME}\usr\bin;${PATH}'` to the `.Renviron` file

```
install.packages(c('devtools'))
```

- Install the *devtools* package from CRAN:

OR

```
install.packages('devtools', lib='~/R/lib')
```

Call the `install_github()` function from the *devtools* package (no need to download the whole package) using the following syntax:

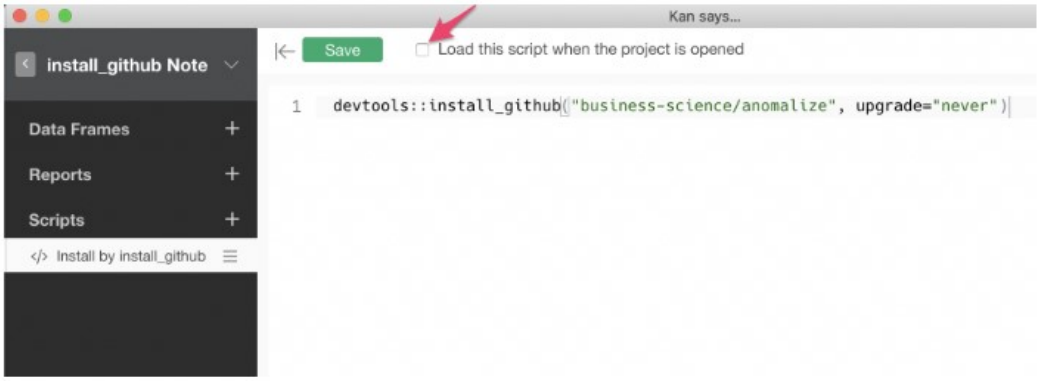
```
devtools::install_github(username/repo_name[/subdir])
```

For example:

OR

```
devtools::install_github('rstudio/shiny')
```

The approach described above is applicable only for public GitHub repositories. To install an R package from a private repository, we need to set the `auth_token` optional parameter of the `install_github()` function with a token obtained from https://github.com/settings/tokens.



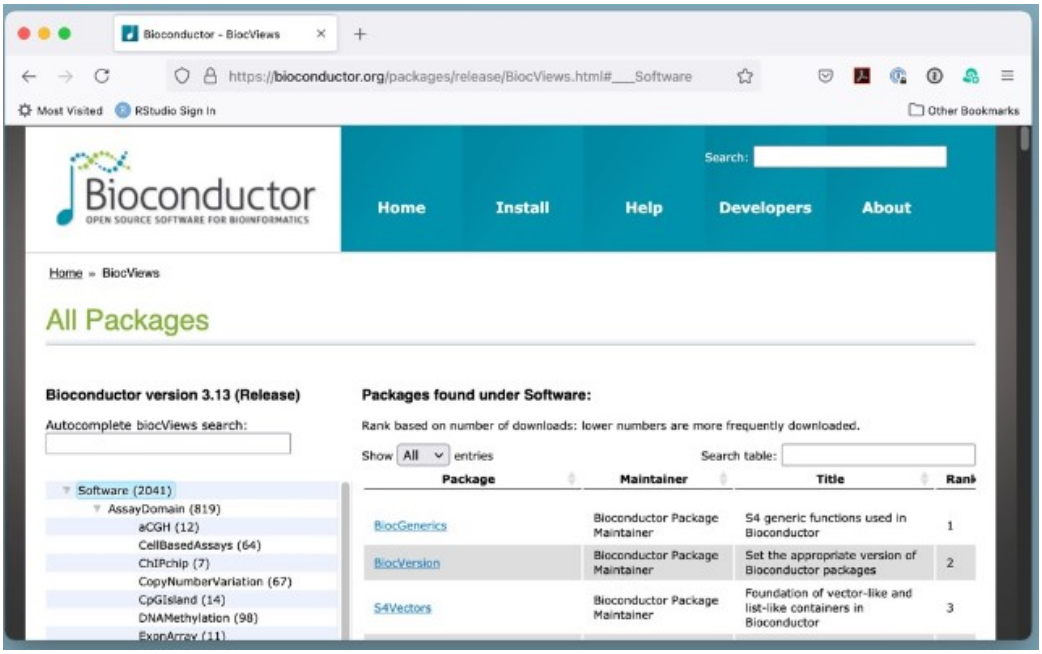
10.3.1 Installing Bioconductor Packages

In the case of Bioconductor, the standard way of installing a package is by first executing the following script:

```
source("https://bioconductor.org/biocLite.R")
```

This will install some basic functions needed to install bioconductor packages, such as the biocLite() function. If you want to install the core packages of Bioconductor, just type it without further arguments:

```
biocLite()
```



If, however, you are interested in just a few particular packages from this repository, you can type their names directly as a character vector:

```
biocLite(c("GenomicFeatures", "AnnotationDbi"))
```

10.4 How to Load Packages

R packages are comprised of a compilation of R functions, data sets, and compiled code that enhance the existing capabilities of R. The packages are placed in the 'library' directory within the R-environment and are produced by the community. An example of a regularly used package in R is "dplyr", which enhances the functionality of working with dataframes. Upon installing R, default packages are automatically included in the local directory 'library' on your workstation. We can load all the default packages by using the code: 'library()'. Prior to utilising it in the code, it is imperative to load the package.

There are two methods to accomplish this:

- Utilising the library() function
- Utilising the need() method.

The most commonly used function to load the package is library(). require() is only used when we have to use the logical values that it returns. It will return TRUE if the package is present and successfully loaded.

10.5 PACKAGE Vs LIBRARY

In R programming, the terms "libraries" and "packages" are often used interchangeably, but there are subtle differences between the two.

10.5.1 Libraries:

General Concept: In programming, a library typically refers to a collection of functions and routines that can be called upon to perform specific tasks or operations.

Usage: Libraries are used to group related functions and data structures together for ease of use and organization within a programming language.

In R: In R, a library can refer to a collection of R functions, data sets, and compiled code stored in a directory. It's a directory where R functions and other extensions are stored.

Loading: In R, libraries are loaded using the library() function, or its shorthand form require(), to make the functions and data sets within the library available for use in the current R session.

10.5.2 Packages:

Specific to R: In R, a package is a specific type of library. It is a collection of R functions, data sets, and compiled code bundled together in a standardized format.

Structure: Packages in R have a specific structure adhering to the guidelines of the Comprehensive R Archive Network (CRAN) or other repositories. This structure includes metadata, documentation, and organized code directories.

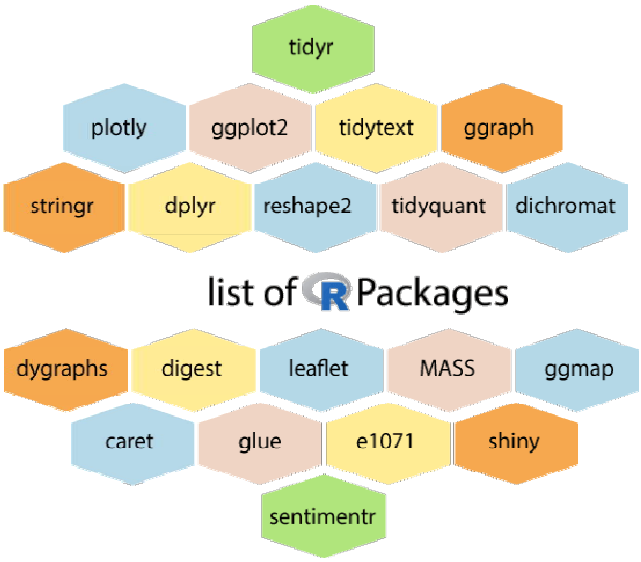
Installation: Packages need to be installed before they can be used. They can be installed from CRAN, GitHub, or other sources using the install.packages() function or devtools package.

Loading: Once installed, packages can be loaded into an R session using the library() or require() function, just like other libraries. Loading a package makes all the functions and datasets within that package available for use.

10.6 IMPORTANT R PACKAGES

There were over 17,000 packages available on the Comprehensive R Archive Network (CRAN), which is the primary repository for R packages. However, the number of packages continues to grow as new packages are developed and existing ones are updated. We will discuss some important packages here.

dplyr: The dplyr package offers a syntax that is both brief and easy to understand for doing data processing tasks. You are able to subset, transform, and summarise data frames in an effective manner thanks to its capabilities, which include filter, select, mutate, and summarise functionality. When it comes to data manipulation, dplyr is an indispensable tool for any data scientist because it allows for the effortless execution of complex operations.



ggplot2 is a strong data visualisation application that is well-known for its plots that are both aesthetically beautiful and customisable within the programme. You are able to build a broad variety of visualisations with ggplot2, which is based on the grammar of graphics. These visualisations range from simple scatter plots to complicated multilayered plots. Because of its adaptability and capacity to manage enormous datasets, ggplot2 is a vital tool for successfully studying and communicating data for scientific purposes.

tidyr is the software that should be used before any other when it comes to reshaping and tidying up data. For the purpose of facilitating the transition of data between wide and long formats, tidyr offers functions such as gather, distribute, and separate. You will be able to effortlessly clean and organise your data with the help of tidyr, ensuring that it is in a structure that is ideal for analysis and visualisation.

One of the most extensive toolkits for machine learning is the caret package, which is an abbreviation for the acronym "Classification And REgression Training." The workflow for tasks such as feature selection, model training, and hyperparameter tuning is simplified as a result of the provision of a unified interface for the construction and evaluation of predictive models. When it comes to applying machine learning algorithms to your data, caret is an invaluable asset to have at your disposal.

randomForest one of the most popular packages for ensemble learning and the construction of decision tree-based models is called randomForest. randomForest is able to handle high-dimensional datasets while still providing reliable and accurate predictions. This is accomplished through the use of random forests, which are a method that aggregates predictions from many decision trees. Problems involving classification and regression are particularly well-suited to its use.

lubridate: Working with dates and times might be difficult, but the lubridate package makes this work much easier to accomplish. The lubridate library provides user-friendly functions that can be used to interpret, manipulate, and format dates and times. Lubridate provides handy tools for effectively managing temporal data, whether you need to extract certain components, calculate time differences, or handle time zones. These tools can be customised to meet your specific needs.

stringr: Are you interested in learning about text mining and string manipulation? A stringr package is the only thing you need to look at. stringr offers a dependable and straightforward user interface for performing operations involving string manipulation. Pattern matching, string extraction, replacement, and other kinds of functions are all included in its capabilities. When working with unstructured text data or carrying out tasks involving text analysis, stringr is an indispensable tool on your machine.

tidy: tidymodels is a collection of packages that were developed with the intention of simplifying the process of machine learning, beginning with the preprocessing stage and ending with the evaluation of the model. Recipes for data preprocessing, parsnip for model formulation, and yardstick for model evaluation are some of the packages that are included in this bundle. Using an approach that is both consistent and tidy, you are able to construct machine learning pipelines with the help of tidymodels.

shiny: shiny is a cutting-edge software that enables you to develop interactive web apps directly from the statistical programming language R. Through the use of shiny, it is possible

to create interactive dashboards, data visualisations, and bespoke tools without the need for substantial expertise of web development. In addition to making your analyses more accessible and interactive, it lets you to share the outcomes of your data science research with other people.

A functional programming toolset that allows for the iteration and manipulation of data structures is provided by the purrr package. You are able to apply operations on lists, data frames, and vectors with the help of its extensive collection of functions, which include map, reduce, filter, and many others. The fact that purrr makes it possible to build data processing workflows that are both efficient and elegant makes it an extremely useful package for managing complex data jobs.

10.7 SUMMARY

In conclusion, this chapter has provided a comprehensive overview of R packages, essential components within the R ecosystem. Starting with the definition of an R package as a collection of functions, datasets, and compiled code, the discussion elucidated the significance of packages in extending R's functionality and facilitating reproducible research. The exploration of R repositories, notably CRAN, highlighted the vast repository of packages available to users for diverse purposes. Moreover, the chapter elucidated the straightforward process of installing packages via the install.packages() function and the subsequent loading of packages into R sessions using library() or require(). Distinguishing between packages and libraries clarified that packages are a specific type of library adhering to standardized structures and conventions. Lastly, an overview of important R packages like dplyr, ggplot2, caret, tidyr, and randomForest showcased their pivotal roles across various domains, underscoring their indispensability in data analysis, visualization, machine learning, and more. Overall, understanding R packages and their utilization is foundational for leveraging the full potential of the R programming language in statistical computing and data science endeavors

10.8 TECHNICAL TERMS

CRAN, Github, Bitconductor, randomForest

10.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. How does CRAN (Comprehensive R Archive Network) function as a repository for R packages?
2. What other repositories exist for R packages, and how do they differ from CRAN?
3. What are the steps involved in installing an R package?
4. What is the distinction between a package and a library in R?
5. What are some important R packages commonly used for data manipulation?

Short questions:

1. What is the purpose of an R package?
2. How are R packages beneficial for R users?
3. What components typically make up an R package?
4. How does the structure of an R package contribute to its functionality?

10.10SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. Crawley, M. J. (2012). The R book. John Wiley & Sons.
3. Albert, J. & Rizzo, M. (2012). R by Example. Springer
4. Teetor, P. (2011). R Cookbook. O'REILLY
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R.Cambridge.

AUTHOR: **Dr. U. Surya Kameswari**

LESSON- 11

INTRODUCTION TO DATA FRAMES

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of DataFrames as a fundamental data structure in R for organizing tabular data.
- Gain proficiency in performing various operations on DataFrames, including subsetting, filtering, sorting, and merging data.
- Practice applying learned concepts through hands-on exercises and real-world examples
- Evaluate and compare different approaches for working with DataFrames

STRUCTURE OF THE LESSION:

11.1 INTRODUCTION

In R, a data frame is a fundamental data structure used for storing and organizing data in a tabular format, similar to a spreadsheet or a database table. It is a two-dimensional array-like structure where each column can contain different types of data, such as numeric, character, factor, or logical. Data frames are commonly used for data manipulation, analysis, and statistical modeling in R.

They provide a structured format for storing data in a tabular manner, making it easier for data scientists to explore, clean, transform, and analyze datasets. Data frames enable data scientists to conduct various data manipulation operations, such as filtering, sorting, and aggregating data, as well as performing complex transformations and computations on columns or rows. Additionally, data frames facilitate the integration of diverse data sources and types, allowing data scientists to combine structured and unstructured data, time series data, spatial data, and more, into a unified framework for analysis.

11.1.1 Features of data frames in R

Rectangular Structure: A data frame has a rectangular structure, meaning that each column must have the same length, but different columns can contain different types of data.

Column Names: Each column in a data frame has a name, which allows you to access the data by column name.

Row Names: Rows in a data frame are indexed with row names, which can be either integers or character strings. By default, row names are set to sequential integers starting from 1.

Mixed Data Types: Data frames can hold mixed data types within columns. For example, one column might contain numeric values, while another might contain strings or factors.

Subsetting: You can subset data frames to extract specific rows or columns based on various conditions using indexing, logical expressions, or column names.

Data Manipulation: Data frames support various operations for data manipulation, such as adding or removing columns, merging with other data frames, reshaping, and transforming data.

Integration with R Ecosystem: Data frames are seamlessly integrated with other R packages and functions commonly used for data analysis and visualization, such as ggplot2, dplyr, tidyr, and others.

Importing and Exporting: R provides functions to import data from external sources, such as CSV files, Excel spreadsheets, databases, and APIs, into data frames. Similarly, you can export data frames to various formats, including CSV, Excel, and R data files.

11.1.2 The role of data frames in data analysis

Data frames serve as a fundamental tool in data analysis, providing a structured and versatile framework for organizing, manipulating, and exploring data. In data analysis tasks, data frames enable analysts to easily import and manage datasets from various sources, such as CSV files, databases, or APIs. Once the data is loaded into a data frame, analysts can perform a wide range of operations, including data cleaning, transformation, and aggregation. For instance, data frames allow analysts to handle missing values, remove duplicates, and convert data types to ensure the quality and consistency of the data. Additionally, data frames support advanced data manipulation techniques, such as merging, reshaping, and pivoting, which are essential for preparing data for analysis and generating insights.

Furthermore, data frames play a crucial role in data exploration and visualization, allowing analysts to gain insights into the underlying patterns and relationships within the data. Analysts can use data frames to compute summary statistics, visualize distributions, and create informative plots and charts to understand the characteristics of the data. With the rich ecosystem of data analysis packages in R, such as ggplot2, dplyr, and tidyr, analysts can seamlessly integrate data frames into their analysis workflow and leverage powerful visualization and statistical techniques to explore complex datasets. By harnessing the capabilities of data frames, analysts can efficiently analyze data, uncover hidden trends and patterns, and derive actionable insights to support decision-making processes in various domains, from business and finance to healthcare and research.

11.1.3 Characteristics of data frame

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

11.2 CREATING A DATA FRAME

A data frame is a two-dimensional data structure which can store data in tabular format.

Data frames have rows and columns and each column can be a different vector. And different vectors can be of different data types.

In R, there are several ways to create a data frame. Here are some common methods:

11.2.1. Using data.frame() Function:

The most straightforward method is to use the 'data.frame()' functionavailable in the base package. We can specify the values for each column directly within the function call.

The syntax of the data.frame() function is

```
dataframe1 <- data.frame
(
first_col  = c(val1, val2, ...),
second_col = c(val1, val2, ...),
   ...
)
```

first_col - a vector with values val1, val2, ... of same data type
second_col - another vector with values val1, val2, ... of same data type and so on

Example:

```
# Creating a data frame using data.frame()
df<- data.frame(
      Name = c("Arya", "Shourya", "Srinivas"),
      Score = c(75, 32, 58),
      Gender = c("Male", "Female", "F"),
Result = c(TRUE, FALSE, TRUE)
    )
   # displaying the data frame
        Name    Age Gender  Married
     1  Arya     25   Male    TRUE
     2  Shourya  30 Female   FALSE
     3  Srinivas 28   Male    TRUE
```

This code creates a data frame named **`df`** with four columns: "Name", "Age", "Gender", and "Married", and three rows of data. Each column contains different types of data, including character, numeric, and logical values.

11.2.2. Combining Vectors or Lists: We can combine vectors or lists of equal lengths into a data frame using the 'data.frame()' function.

Creating vectors

```
names<- c("Arya", "Shourya", "Srinivas")
scores<- c(25, 30, 28)
genders<- c("Male", "Female", "Male")
results<- c(TRUE, FALSE, TRUE)
```

Combining vectors into a data frame

```
df<- data.frame(Name = names, Age = ages, Gender = genders, Score = scores)
```

11.2.3. Reading Data from External Files: We can read data from external files, such as CSV, Excel, or text files, into a data frame using functions like 'read.csv()', 'read.table()', or specialized functions from other packages.

Reading data from a CSV file

```
df<- read.csv("data.csv")
```

11.2.4. Converting Other Data Structures: We can convert other data structures, such as matrices or lists, into data frames using functions like 'as.data.frame()'.

```
# Creating a matrix

mat<- matrix(1:12, nrow = 4)

# Converting matrix to data frame

df<- as.data.frame(mat)
```

11.2.5. Generating Data with Functions: We can generate data using functions like 'seq()', 'rep()', or 'rnorm()' and then combine them into a data frame.

```
# generating data

names<- c("Arya", "Shourya", "Srinivas")
ages<- seq(34, 52, by = 1)
genders<- rep(c("Male", "Female"), length.out = length(names))
scores<- sample(c(TRUE, FALSE), size = length(names), replace = TRUE)


# combining data into a data frame
```

```
df<- data.frame(Name = names, Age = ages, Gender = genders, Score = scores)
```

These methods offer flexibility in creating data frames in R, allowing us to choose the most suitable approach based on our data source and requirements.

11.3 OPERATIONS ON DATA FRAME

In R, there are various operations that can be performed on data frames to manipulate, analyze, and visualize data. Here are some common operations on data frames:

11.3.1. Subsetting Data: Subsetting allows you to extract specific rows or columns from a data frame based on certain conditions. We can subset data frames using indexing, logical expressions, or column names. We can use [ ], [[ ]], or $ to access specific column of a data frame

```
# Selecting specific columns
selected_columns<- df[, c("Name", "Age")]

# Selecting rows based on a condition
subset_data<- df[df$Age> 25,]

# pass index number inside [ ]
print(dataframe1[1])

# pass column name inside [[ ]]
print(dataframe1[["Name"]])

# use $ operator and column name
print(dataframe1$Name)

Output:
1 Arya
2 Shourya
3 Nivas
[1] "Arya", "Shourya", "Srinivas"
[1] "Arya", "Shourya", "Srinivas"
```

11.3.2. Filtering Data:

Filtering data frames in R involves selecting specific rows that meet certain conditions. There are several ways to filter data frames, including using indexing, logical conditions, or functions like subset() or dplyr package functions.

```
# Filtering data based on a condition
filtered_data<- subset(df, Age > 25 & Result== TRUE)

# Filtering data based on a condition using indexing
```

```
filtered_data<- df[df$Age> 25 &df$Result == TRUE, ]

# Filtering data based on a condition using logical expressions
filtered_data<- df[df$Age> 25 &df$Result == TRUE, ]

# Filtering data using subset() function
filtered_data<- subset(df, Age > 25 & Result == TRUE)

# Filtering data using dplyr package
library(dplyr)
filtered_data<- filter(df, Age > 25, Result == TRUE)
```

11.3.3. Adding Columns:
Adding new columns to a data frame in R can be done using various methods.

We can directly assign values to a new column by referencing the data frame with the new column name.

```
  # Adding a new column using assignment
df$New_Column<- c("Value1", "Value2", "Value3")
```

The mutate()function from the **`dplyr`** package allows you to add new columns while preserving the original data frame.

```
# Adding a new column using mutate() function
library(dplyr)
df<- mutate(df, New_Column = c("Value1", "Value2", "Value3"))
```

We can combine the existing data frame with a vector or matrix to add a new column using the cbind() function.

```
# Adding a new column using cbind()
new_column<- c("value1", "value2", "value3")
df<- cbind(df, new_column)
```

The transform() function allows you to add one or more new columns to a data frame.

```
# Adding a new column using transform()
df<- transform(df, new_column = c("value1", "value2", "value3"))
```

11.3.4. Removing Columns: We can remove columns from a data frame using indexing or the 'subset()' function.
Removing columns from a data frame in R can be achieved using several methods. Here are some common approaches:

Example dataframe:

```
df<- data.frame(Colors = c("red", "green", "blue", "yellow", "orange"),
                Shapes = c("square", "circle", "triangle", "rectangle", "cone"),
                Sizes = c("small", "medium", "large", "small", "medium"),
                Length = c(10, 25, 150, 5, 45)
               )
print(df)
```

Output:

```
  Colors    Shapes   Sizes  Length
1    red    square   small      10
2  green    circle  medium      25
3   blue  triangle   large     150
4 yellow rectangle   small       5
5 orange      cone  medium      45
```

1. Using Indexing:

We can remove columns by indexing the columns you want to keep.

```
   # Removing columns using indexing
df<- df[, -c(2, 4)]  # Removes 2nd and 4th columns
```

example:

```
      df<- subset(df, select = -2)
```

it removes Shapes column from the dataframedf.

2. Using Negative Indexing:

We can use negative indexing to exclude specific columns.

```
   # Removing columns using negative indexing
df<- df[, -c(2, 4)]  # Removes 2nd and 4th columns
```

3. Using Subset() Function:

The 'subset()' function can be used to select columns that you want to keep.

```
   # Removing columns using subset() function  single column
df<- subset(df, select = -column_name)

   # Removing columns using subset() function
df<- subset(df, select = -c(Column2, Column4))
```

Examples: To remove the "Shapes" column using subset():

```
      df<- subset(df, select = -Shapes)
```

to remove multiple columns:

```
      df<- subset(df, select = -c(Shapes, Sizes))
```

4. Using Dplyr Package:

In R, the dplyr package is one of the most popular package for data manipulation. It makes data wrangling easy. You can install package by using the command below -

```
      install.packages("dplyr")
```

The 'dplyr' package provides a concise syntax for data manipulation tasks. We can use the 'select()' function to remove columns.

```
# Removing columns using dplyr package
library(dplyr)
df<- select(df, -Column2, -Column4)
```

5. Using drop() Function:
Base R provides functions like 'drop()' and 'subset()' to remove columns.

```
# Using drop() function
df<- df[, -which(names(df) %in% c("Column2", "Column4"))]
```

6. Using the '$' Operator:
We can remove columns directly by referencing them with the '$' operator.

```
# Removing columns using $
df$Column2 <- NULL
df$Column4 <- NULL
```

Choose the method that fits your preference and integrates well with your code or workflow. All these methods will remove the specified columns from the data frame 'df'.

6. Merging Data:

Merging data frames in R involves combining two or more data frames based on a common column or key. There are several functions and approaches available for merging data frames in R. Here are some common methods along with examples:

1. Using 'merge()' Function:
The 'merge()' function in base R merges two data frames based on common column(s).

```
# Merging data frames using merge() function
merged_df<- merge(df1, df2, by = "ID")
```

This will merge 'df1' and 'df2' based on the common column "ID".

2. Using 'rbind()' Function:
If the data frames have the same structure (same columns), you can stack them vertically using the 'rbind()' function.

```
# Merging data frames vertically using rbind() function
merged_df<- rbind(df1, df2)
```

This will concatenate 'df1' and 'df2' rows-wise.

3. Using 'bind_cols()' Function from 'dplyr' Package:
The 'bind_cols()' function from the 'dplyr' package can be used to combine data frames column-wise.

```
# Merging data frames column-wise using bind_cols() function
library(dplyr)
merged_df<- bind_cols(df1, df2)
```

This will concatenate 'df1' and 'df2' side by side.

4. Using Joins with 'dplyr' Package:

The ‘dplyr’ package provides functions like ‘inner_join()’, ‘left_join()’, ‘right_join()’, and ‘full_join()’ to perform different types of joins between data frames.

```
# performing a left join using dplyr
library(dplyr)
merged_df<- left_join(df1, df2, by = "ID")
```

This will perform a left join between ‘df1’ and ‘df2’ based on the common column "ID".

11.4 EXPANDING A DATA FRAME

Expanding a data frame in R typically refers to adding more rows or columns to an existing data frame. Here are ways to expand a data frame in both dimensions:

1. Adding Rows:

To add more rows to an existing data frame, you can use functions like ‘rbind()’, ‘bind_rows()’ from the ‘dplyr’ package, or simply indexing with the ‘[nrow()+1,]’ notation.

```
# Using rbind()
new_rows<- data.frame(Name = c("Shourya", "Arya"), Age = c(35, 40))
df<- rbind(df, new_rows)
```

```
# Using bind_rows() from dplyr
library(dplyr)
new_rows<- data.frame(Name = c("Shourya", "Arya"), Age = c(35, 40))
df<- bind_rows(df, new_rows)
```

```
# Using indexing
df[nrow(df) + 1, ] <- c("Shourya", 35)  # Add a new row at the end
```

2. Adding Columns:

To add more columns to an existing data frame, you can use functions like ‘cbind()’ or ‘bind_cols()’ from the ‘dplyr’ package.

```
# Using cbind()
new_columns<- data.frame(Salary = c(50000, 60000))
df<- cbind(df, new_columns)
```

```
# Using bind_cols() from dplyr
library(dplyr)
new_columns<- data.frame(Salary = c(50000, 60000))
```

```
df<- bind_cols(df, new_columns)
```

These methods allow us to expand our data frame either by adding more rows or more columns, depending on your data expansion needs. Remember to ensure that the dimensions of the data being added match the existing data frame's structure.

11.5 APPLYING FUNCTIONS TO DATA FRAME

Some common functions used with DataFrames in R:

To understand these functions very well, here we are taking the following data frame of the admission data of the following six students to a specific programme of ANU taken as an example.

#Creating and assigning a data frame

```
>df<- data.frame(
+ c("Srinivas","Surya","Shourya","Arya","Reddy","Sravani"),
+ as.factor(c("Male", "Male", "Male", "Female", "Male","Female")),
+ c(78,45,83,75,89,62),
+ c(TRUE, FALSE,TRUE, FALSE, TRUE, TRUE))
```

#Setting the column names

```
>names(df) <- c("Name","Gender","Score","Result")
```

#Printing the data frame

```
>print(df)
```

	Name	Gender	Score	Result
1	Srinivas	Male	78	TRUE
2	Surya	Male	45	FALSE
3	Shourya	Male	83	TRUE
4	Arya	Female	75	FALSE
5	Reddy	Male	89	TRUE
6	Srvani	Female	62	TRUE

11.5.1 str()

#Internal structure of the data frame

```
>str(Adm.data)
'data.frame': 6 obs. of 4 variables:
$ Name :chr ("Srinivas","Surya","Shourya","Arya", …
$ Gender : Factor w/ 2 levels "Female","Male": 2 2 2 1 2 1
$ Score: num 78, 45, 83, 75, 89, …
$  Result: logi TRUE FALSE FALSEFALSE TRUE FALSE
```

From the internal structure, it is clear that the obtained information consists of the complete details on the 6 observations of 4 columns or variables (whose names are written after '$' operator. The output depicts that the Name variable is a character variable (as it is specified by chr), the Gender variable is a factor variable with two levels (as it is specified by Factor), the Score variable is a numeric variable (as it is specified by num) and the last variable Result is a logical variable (as it is specified by logi).

11.5.2. head() and tail():

The first argument of the head() and tail() functions is assigned as the name of the data frame and the second argument n is assigned as the number rows to be viewed. For the illustration purpose, we now supply df data frame as an argument to the head() and tail() functions to get default number of rows and specified number of rows as follows:

#Getting first six rows (by default)

```
>head(df)
      Name  Gender     Score Result
1     Srinivas   Male  78    TRUE
2     Surya Male  45    FALSE
3     Shourya    Male  83    TRUE
4     Arya  Female     75    FALSE
5     Reddy Male  89    TRUE
6     Srvani     Female     62    TRUE
```

Getting or viewing the first two rows

```
>head(df,2)
      Name  Gender     Score Result
1     Srinivas   Male  78    TRUE
2     Surya Male  45    FALSE
```

Getting last six rows (by default)

```
>tail(df)
      Name  Gender     Score Result
1     Srinivas   Male  78    TRUE
2     Surya Male  45    FALSE
3     Shourya    Male  83    TRUE
4     Arya  Female     75    FALSE
5     Reddy Male  89    TRUE
6     Srvani     Female     62    TRUE
```

Getting last three rows (by default)

```
>tail(df,3)
      Name  Gender     Score Result
```

```
4     Arya  Female     75    FALSE
5     Reddy Male  89    TRUE
6     Srvani      Female     62    TRUE
```

11.5.3 Summary()

This function provides summary statistics of the data in a DataFrame.

Syntax: summary(data_frame_name)

This function gives us summary of 6 numbers, namely, minimum, 1st quartile, median, mean, 3rd quartile and maximum for the numeric or integer type data. The length, class and mode for the character type data. Mode and count of TRUE and FALSE for logical type data. Count of levels for the factor type data. Observe that the summary() function also gives the count of NA's. To verify it, we now supply the `df` data frame to get the summary on each column of `df` data frame as follows:

#Getting summary of each column of the data frame

```
>summary(df)
 Name Gender Score Result
Length:6 Female:2 Min. :65.04 Mode :logical
Class :character Male :4 1st Qu.:75.22 FALSE:4
Mode :character Median :80.13 TRUE :2
Mean :78.85
 3rd Qu.:85.31
 Max. :88.55
NA's :0
```

Hence, the obtained output confirms the aforementioned statements. Since the Name variable is of character type, the Gender variable is of factor type, the score variable is of numeric type and the result variable is of logical type.

11.5.4 dim():

The dimension of R objects like matrices and data frames can be obtained using the dim() function. This function returns the dimensions (number of rows and columns) of a DataFrame. It returns the number of rows and columns of a matrix.

#Getting the dimensions of a data frame

```
>dim(df)
[1] 6 4
```

11.5.5. names()

Names of the R objects can be set using the names() function available in the base package. Setting names are very useful for writing self-describing and readable code. When this function is used alone on an R object, it will return the names of the R object. Note that the names() function accepts different objects as argument such as vector, matrix, list and data frames

Example: Assign name to an object

```
# Creating a vector
x <- c(23001, 23002, 23003, 23004, 23005)

# Assigning names using names() function
names(x) <- c("Vijaya", "Lakshmi", "Bala", "Namitha", "Gowri")

# Printing name vector that is assigned
names(x)
  [1] "Vijaya", "Lakshmi", "Bala", "Namitha", "Gowri"
# Printing updated vector
print(x)
Vijaya, Lakshmi, Bala,Namitha, Gowri
23001, 23002, 23003, 23004, 23005
```

11.5.6 nrow() and ncol

The nrow() and ncol() functions find the number of rows and columns of a given data frame respectively.

Both the nrow() and ncol() functions take the same parameter value. This value represents the data frame object whose number of rows and columns is to be determined.

```
# number of rows
nrow(df)
[1]  6

# number of columns
ncol(df)
[1]  4
```

11.5.7 cbind() and rbind():

In R, while doing data manipulation and analysis, one might need to combine vectors and data frames to create a holistic data set. The cbind() and rbind() are powerful functions, allowing us to combine vectors and data frames efficiently.

Example:

Let's start by creating two vectors, having first and last names of students, by using following commands.

```
f_name<- c("Ganga","Satya","Ramesh")
l_name<- c("kancharla","Ravi","Inampudi")
```

cbind() which is essentially stands for column bind. This function is used to bind vectors or matrices as columns to create a new matrix. So for the above vectors we just created, if we want to combine them by column, the following command is used

```
full_names<- cbind(f_name,l_name)
print(full_names)

     f_name     l_name
1    Ganga Kancharla
2    Satya Ravi
3    Ramesh     Inampudi
```

rbind() combines the rows of vectors or data frames. It binds vectors, matrices, or data frames by rows to create a new vector, matrix, or data frame.

```
full_namesr<- rbind(f_name,l_name)
print(full_namesr)

           V1         V2         V3
f_name     Ganga      Satya      Ramesh
L_name     Kancharla  Ravi       Inampudi
```

11.6 SUMMARY

The chapter begins with an introduction to DataFrames, a fundamental data structure in R for organizing and manipulating data. It covers the creation of DataFrames using the `data.frame()` function and explores various operations that can be performed on DataFrames, such as subsetting, filtering, and merging. The chapter also delves into expanding DataFrames by adding new columns or rows and applying functions to DataFrames to perform calculations or transformations on the data. Through practical examples and explanations, readers gain a comprehensive understanding of how to effectively work with DataFrames in R, from basic creation to advanced data manipulation techniques.

11.7 TECHNICAL TERMS

Data frame, R Eco system, data analysis,

11.11 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the process of subsetting a DataFrame based on specific criteria. Give an example.
2. Describe the process of expanding a DataFrame by adding new columns or rows. Give examples of each.
3. Can you demonstrate how to apply a custom function to a DataFrame using the `apply()` function? Provide an example.
4. Explain the concept of vectorized functions and how they can be applied to DataFrames for efficient computation.

Short Questions:

1. What is a DataFrame and how does it differ from other data structures in R?
2. What are some common operations that can be performed on a DataFrame, such as subsetting and filtering?
3. How do you merge two DataFrames in R? Provide an example using the `merge()` function.
4. What are some methods for applying functions to a DataFrame in R? How do these methods differ?

11.12 SUGGESTED READINGS

1. SeemaAcharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team
3. Crawley, M. J. (2012). The R book. John Wiley & Sons.
4. Albert, J. & Rizzo, M. (2012). R by Example. Springer
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R.Cambridge.

AUTHOR: **Mrs. A Sarvani**

LESSON- 12

READING AND GETTING DATA INTO R INTERNAL DATA

OBJECTIVES:

After going through this lesson, you will be able to

- How to read data from various file formats commonly encountered in data analysis tasks, such as CSV files, tab-separated value files, and Excel files
- Proficient in using R functions like read.csv(), read.table(), and read_excel() to import data into R and create data frames
- Equipped to handle real-world datasets and perform meaningful analyses using R programming.
- Develop the ability to choose the appropriate file reading function and specify relevant parameters based on the format and structure of the data

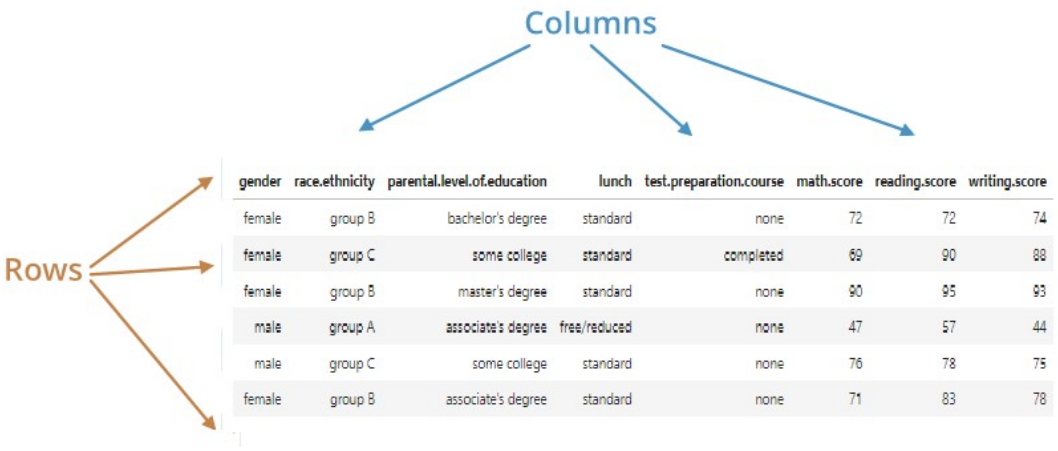
STRUCTURE OF THE LESSION:

12.1 INTRODUCTION

Reading and importing data into R from the local system is a crucial skill for any data analyst or scientist. It serves as the initial step in the data analysis pipeline, allowing researchers to access, explore, and manipulate datasets for further investigation. Whether the data resides in a CSV file, Excel spreadsheet, text file, or other formats, being able to seamlessly import it into R provides analysts with the flexibility and control needed to conduct comprehensive analyses. By mastering this process, practitioners can harness the power of R's robust statistical and graphical capabilities to derive valuable insights and make informed decisions.

Understanding how to read data into R from the local system is essential due to the diverse sources and formats in which data is stored. Real-world datasets often come from various sources such as research studies, surveys, government databases, and corporate systems, each with its own structure and encoding. Consequently, knowing how to handle different file types, handle missing values, and preprocess data is critical for ensuring data quality and integrity. Moreover, the ability to import data efficiently enables analysts to streamline their workflow and focus on exploring and analyzing data rather than spending excessive time on data wrangling tasks.

Furthermore, the need for reading and importing data into R extends beyond simple file manipulation; it encompasses the integration of data from multiple sources and the automation of data pipelines. Analysts often work with complex datasets spanning multiple files or databases, requiring them to merge, join, and clean data before analysis. Additionally, automating the data import process using scripts or functions allows analysts to maintain reproducibility, scalability, and efficiency in their work. Consequently, mastering the techniques for reading and importing data into R empowers analysts to tackle real-world data challenges effectively and unlock the full potential of their analyses.



gender	race.ethnicity	parental.level.of.education	lunch	test.preparation.course	math.score	reading.score	writing.score
female	group B	bachelor's degree	standard	none	72	72	74
female	group C	some college	standard	completed	69	90	88
female	group B	master's degree	standard	none	90	95	93
male	group A	associate's degree	free/reduced	none	47	57	44
male	group C	some college	standard	none	76	78	75
female	group B	associate's degree	standard	none	71	83	78

12.2 LOAD DATA FRAMES

Loading data frames in R serves as the foundation for data analysis, enabling analysts to access and manipulate structured data efficiently. Data frames, being two-dimensional data structures, are particularly well-suited for representing tabular data, where rows correspond to observations and columns represent variables or attributes. By loading data frames into R, analysts gain the ability to perform a wide range of data manipulation tasks, including data cleaning, transformation, summarization, and visualization. This capability is essential for exploring and understanding the underlying patterns and relationships within the data, ultimately leading to informed decision-making and actionable insights.

Moreover, loading data frames in R facilitates seamless integration with the vast ecosystem of statistical and graphical packages available in the R programming language. Once data is loaded into R, analysts can leverage specialized functions and techniques provided by these packages to conduct advanced analyses and visualizations. Whether it's fitting statistical models, performing hypothesis tests, or creating sophisticated plots, having data in the form of data frames allows analysts to apply these techniques directly to their datasets. This integration fosters a highly efficient and versatile analytical workflow, empowering analysts to tackle diverse data analysis tasks with ease and confidence.

In R, you can read data from files outside of the R environment. One may also write data to files that the operating system can store and further access. There is a wide range of file formats, including CSV, Excel, binary, and XML, etc., R can read and write from.

12.3 READINGFROM .CSV FILES,

12.3.1 Input as CSV File

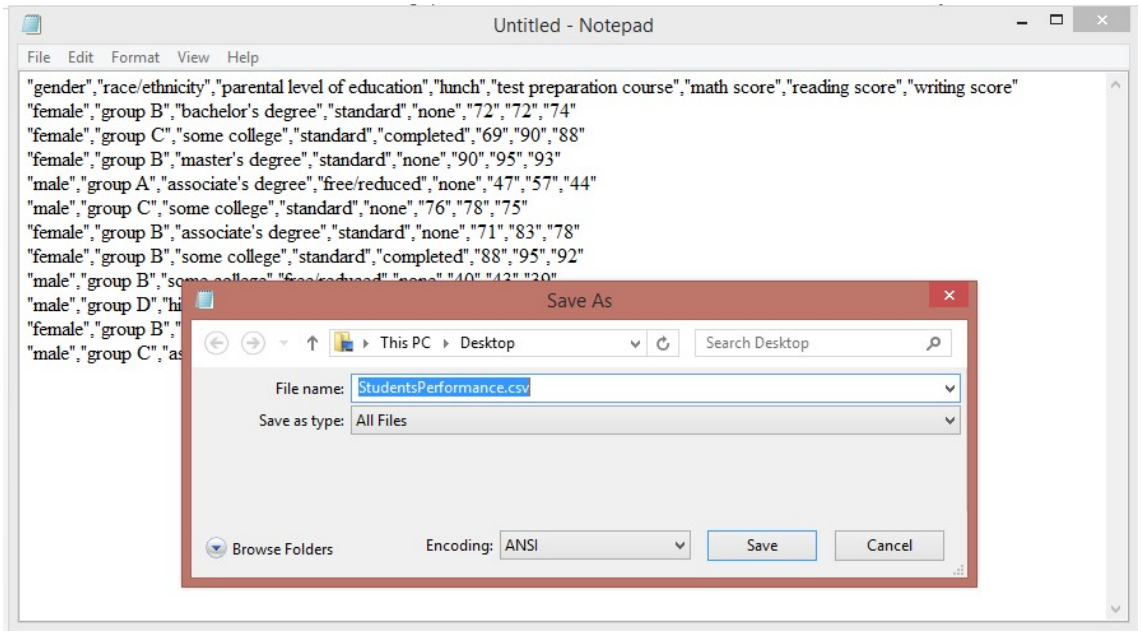
A CSV (Comma-Separated Values) file is a plain text file format commonly used for storing tabular data. In a CSV file, each line represents a row of data, and the values within each row are separated by commas (or other delimiters like tabs or semicolons). The first row often contains column headers, providing names for each column in the dataset.

CSV files are widely used for exchanging data between different software applications because they are simple, lightweight, and can be easily read and written by both humans and machines. Additionally, CSV files can be opened and edited using spreadsheet software like Microsoft Excel, making them accessible to a wide range of users. Due to their simplicity and versatility, CSV files are a popular choice for storing and sharing structured data in various fields, including data analysis, research, and database management.

To create a CSV (Comma-Separated Values) file, you can use a text editor or spreadsheet software. Here's how you can create a CSV file using both methods:

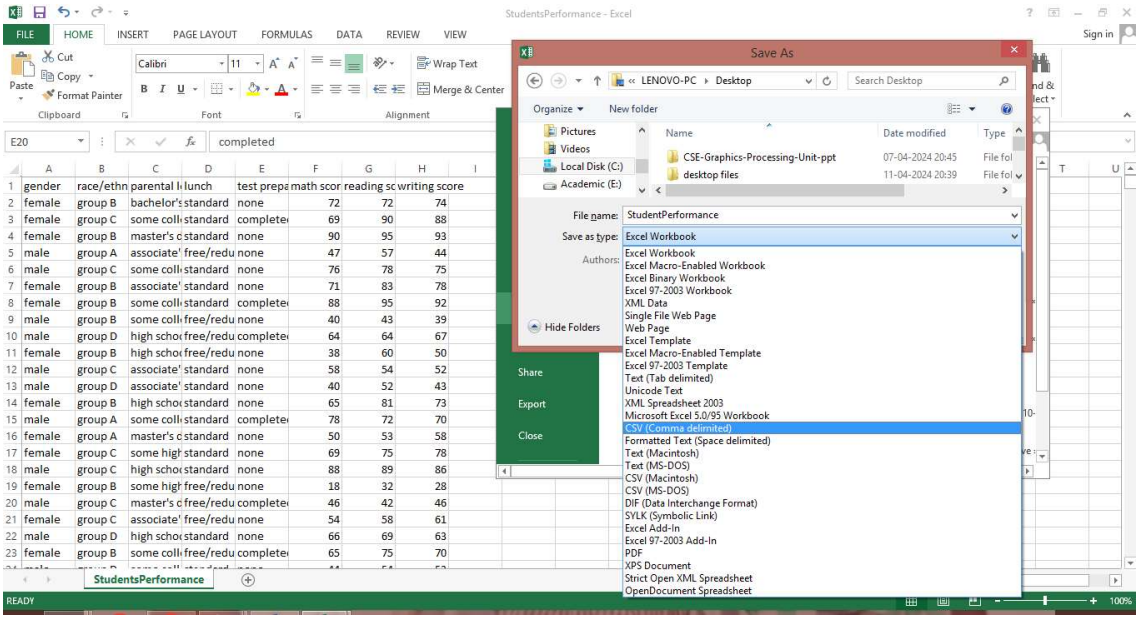
1. Using a Text Editor:

- Open a text editor such as Notepad (Windows), TextEdit (Mac), or any other text editor of your choice.
- Enter your data in rows and columns, separating each value with a comma.
- Optionally, you can include a header row with column names.
- Save the file with a '.csv' extension. For example, 'StudentsPerformance.csv'.



2. Using Spreadsheet Software (e.g., Microsoft Excel)

- Open your spreadsheet software (e.g., Microsoft Excel).
- Enter your data into cells in rows and columns.
- Optionally, you can include a header row with column names.
- Once you have entered your data, go to the "File" menu and select "Save As".
- Choose the CSV file format from the available options.
- Save the file with a '.csv' extension. For example, 'data.csv'.



After creating the CSV file, you can use it to import data into R or any other software application that supports CSV files.

12.3.2 Reading a CSV File

The `read.sv()` function is used to read data from a CSV (Comma-Separated Values) file and create a data frame. It is part of the base R package and is commonly used for importing tabular data stored in CSV format.

Syntax:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"", ...)
```

file: The path to the CSV file you want to read.
header: A logical value indicating whether the first row of the CSV file contains column names. By default, it is set to TRUE.
sep: The character used to separate values in the CSV file. By default, it is set to , (comma).
quote: The character used to quote strings in the CSV file. By default, it is set to " (double quote).
...: Additional arguments passed to read.csv().

After reading the CSV file into R using read.csv(), you will have a data frame (df in the example) containing the data from the CSV file, which you can then analyze, manipulate, and visualize using various R functions and packages.

Read data from a CSV file named "StudentPerformance.csv" in the current directory

```
data<- read.csv("StudentsPerformance.csv")
```

View the structure of the data frame

```
str(df)
```

```
'data.frame':   1000 obs. of  8 variables:
 $ gender                     : Factor w/ 2 levels "female","male":
1 1 1 2 2 1 1 2 2 1 ...
 $ race.ethnicity             : Factor w/ 5 levels "group A","group
B",..: 2 3 2 1 3 2 2 2 4 2 ...
 $  parental.level.of.education:  Factor  w/  6  levels  "associate's
degree",..: 2 5 4 1 5 1 5 5 3 3 ...
x  $  lunch                                  :  Factor  w/  2  levels
"free/reduced",..: 2 2 2 1 2 2 2 1 1 1 ...
 $   test.preparation.course            :   Factor   w/   2   levels
"completed","none": 2 1 2 2 2 2 1 2 1 2 ...
 $ math.score                 : int  72 69 90 47 76 71 88 40 64 38
...
 $ reading.score              : int  72 90 95 57 78 83 95 43 64 60
...
 $ writing.score             : int  74 88 93 44 75 78 92 39 67 50.
```

View the first few rows of the data frame

```
head(df)
```

gender	race.ethnicity	parental.level.of.education	lunch	test.preparation.course	math.score	reading.score	writing.score
female	group B	bachelor's degree	standard	none	72	72	74
female	group C	some college	standard	completed	69	90	88
female	group B	master's degree	standard	none	90	95	93
male	group A	associate's degree	free/reduced	none	47	57	44
male	group C	some college	standard	none	76	78	75
female	group B	associate's degree	standard	none	71	83	78

12.4 READING FROM TAB SEPARATED VALUE FILES

12.4.1 Input as TSV File

Tab-separated value (TSV) files are a type of text file used for storing structured data in a tabular format. In TSV files, each line represents a row of data, and the values within each row are separated by tab characters ("\t"). TSV files are similar to comma-separated value (CSV) files, but instead of using commas as delimiters, they utilize tabs. This format is particularly useful when dealing with data that may contain commas within the values themselves, as using tabs as separators avoids potential parsing issues. TSV files are widely used for data interchange and storage in various fields, including data analysis, bioinformatics, and database management, due to their simplicity, ease of creation, and compatibility with text-based editors and spreadsheet software.

One of the advantages of TSV files is their compatibility with a wide range of software applications and programming languages. Since TSV files are essentially plain text files with tab-delimited values, they can be easily opened and edited using text editors like Notepad (Windows), TextEdit (Mac), or any other text editor. Additionally, TSV files can be imported into spreadsheet software such as Microsoft Excel, Google Sheets, or LibreOfficeCalc, where the tab-separated values are automatically parsed into rows and columns, facilitating data visualization and manipulation. Furthermore, TSV files can be processed programmatically using programming languages like R, Python, or Java, where they can be read, manipulated, and analyzed using specialized libraries and tools, making them a versatile and widely supported format for storing and exchanging structured data.

To read data from a .tsv (tab-separated values) file in R, you can use the `read.table()` function, specifying the `sep` parameter as `"\t"` to indicate tab separation.

1. Using a Text Editor:

- Open a text editor such as Notepad (Windows), TextEdit (Mac), or any other text editor of your choice.
- Enter your data in rows and columns, separating each value with a tab character (`\t`).
- Optionally, you can include a header row with column names.
- Save the file with a `.tsv`extension. For example, 'StudentPerformance.tsv'.

2. Using Spreadsheet Software (e.g., Microsoft Excel)

- Open your spreadsheet software (e.g., Microsoft Excel).
- Enter your data into cells in rows and columns.
- Optionally, you can include a header row with column names.
- Once you have entered your data, go to the "File" menu and select "Save As".
- Choose the TSV file format from the available options.
- Save the file with a '.tsv' extension. For example, 'StudentPerformance.tsv'.

After creating the TSV file, you can use it to import data into R or any other software application that supports TSV files.

12.4.2 Reading a TSV File

Syntax:

```
read.table(file, header = FALSE, sep = "\t", ...)
```

file: The path to the .tsv file you want to read.
header: A logical value indicating whether the first row of the .tsv file contains column names. By default, it is set to FALSE.
sep: The character used to separate values in the .tsv file. In the case of .tsv files, it should be set to "\t" to indicate tab separation.
...: Additional arguments passed to read.table().

After executing the above code, you will have a data frame (df in the example) containing the data from the .tsv file, which you can then analyze, manipulate, and visualize using various R functions and packages.

Read data from a TSV file named "StudentPerformance.tsv" in the current directory

```
df<- read.table("StudentPerformance.tsv", header = TRUE, sep = "\t")
```

View the structure of the data frame

```
str(df)
```

```
'data.frame':   1000 obs. of  8 variables:
 $ gender                     : Factor w/ 2 levels "female","male": 1 1 1 2 2 1 1 2 2 1 ...
 $ race.ethnicity             : Factor w/ 5 levels "group A","group B",..: 2 3 2 1 3 2 2 2 4 2 ...
 $ parental.level.of.education: Factor w/ 6 levels "associate's degree",..: 2 5 4 1 5 1 5 5 3 3 ...
 $ lunch                      : Factor w/ 2 levels "free/reduced",..: 2 2 2 1 2 2 2 1 1 1 ...
 $ test.preparation.course    : Factor w/ 2 levels "completed","none": 2 1 2 2 2 2 1 2 1 2 ...
 $ math.score                 : int  72 69 90 47 76 71 88 40 64 38 ...
 $ reading.score              : int  72 90 95 57 78 83 95 43 64 60 ...
 $ writing.score              : int  74 88 93 44 75 78 92 39 67 50 ...
```

View the first few rows of the data frame

```
head(df)
```

gender	race.ethnicity	parental.level.of.education	lunch	test.preparation.course	math.score	reading.score	writing.score
female	group B	bachelor's degree	standard	none	72	72	74
female	group C	some college	standard	completed	69	90	88
female	group B	master's degree	standard	none	90	95	93
male	group A	associate's degree	free/reduced	none	47	57	44
male	group C	some college	standard	none	76	78	75
female	group B	associate's degree	standard	none	71	83	78

12.5 READING FROM EXCEL

12.5.1 Input as EXCEL File

An Excel file is a digital spreadsheet created and maintained using Microsoft Excel, a widely used spreadsheet software application. These files are commonly employed for organizing, analyzing, and presenting tabular data in a structured format. Excel files typically consist of

one or more worksheets, each containing a grid of cells arranged in rows and columns. Users can enter data into these cells, which can include text, numbers, dates, or formulas, allowing for various calculations and manipulations.

Excel files offer a multitude of features, including built-in formulas and functions, extensive formatting options, and charting tools. Users can utilize formulas and functions to perform calculations, manipulate data, and automate tasks within the spreadsheet. Furthermore, Excel provides a wide range of formatting options for customizing the appearance of data, such as fonts, colors, borders, and cell styles. Additionally, users can create visually appealing charts and graphs to visualize data trends and relationships, enhancing the clarity and impact of their presentations. Excel files are compatible with various software applications and platforms, enabling seamless data interchange and collaboration across different environments.

12.5.2 Reading a EXCEL File

To read data from an Excel file (.xlsx) into R, you can use the 'read_excel()' function from the 'readxl' package. Here's how you can do it:

First, make sure you have the 'readxl' package installed. If not, you can install it using:

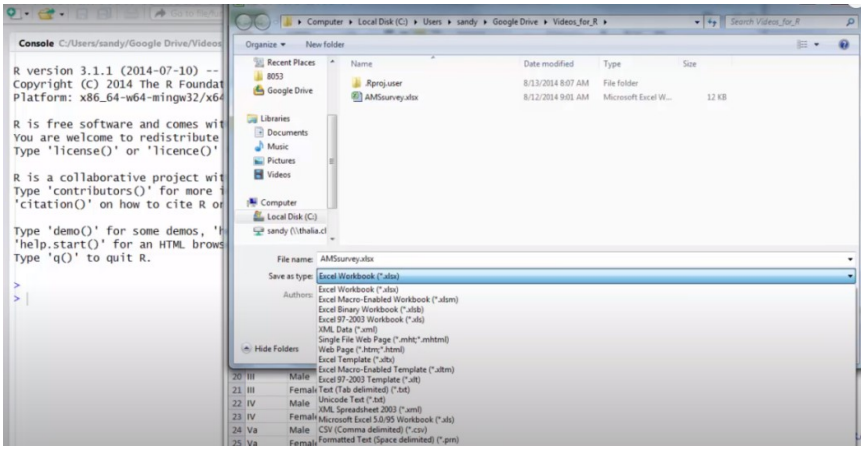
```
install.packages("readxl")
```

Once the package is installed, you can use the 'read_excel()' function to read data from an Excel file:

```
library(readxl)
```

- Specify the path to your Excel file

```
excel_file<- "path/to/your/file.xlsx"
```



- Read data from the Excel file into a data frame

```
df<- read_excel(excel_file)
```

- View the structure of the data frame

```
str(df)
```

- View the first few rows of the data frame

```
head(df)
```

Replace '"path/to/your/file.xlsx"' with the actual path to your Excel file. The 'read_excel()' function automatically detects the sheet name and reads the data from the first sheet by default. If you want to read data from a specific sheet, you can specify the sheet name or index using the 'sheet' parameter:

- Read data from a specific sheet named "Sheet2"

```
df<- read_excel(excel_file, sheet = "Sheet2")
```

Or, you can specify the sheet index (1-based) instead of the sheet name:

- Read data from the second sheet

```
df<- read_excel(excel_file, sheet = 2)
```

After executing the 'read_excel()' function, you will have a data frame ('df' in the examples) containing the data from the Excel file, which you can then analyze, manipulate, and visualize using various R functions and packages.

1.6 SUMMARY

In the chapter on R programming covering Introduction, Load data frames, and Reading from .CSV files, tab-separated value files, and Excel files, students are introduced to fundamental concepts and techniques for handling and manipulating tabular data in R. They start by understanding the basics of the R programming language and its capabilities for data analysis. They learn how to create, manipulate, and explore data frames, which serve as the primary data structure for organizing tabular data. Additionally, students acquire essential skills for importing data from external sources, including CSV files, tab-separated value files, and Excel files, using functions like ‘read.csv()’, ‘read.table()’, and ‘read_excel()’. Through practical examples and exercises, students gain proficiency in reading data into R, which forms the foundation for subsequent data analysis tasks. This chapter equips students with the necessary knowledge and skills to effectively handle and analyze tabular data in R, empowering them to tackle real-world data analysis challenges with confidence and efficiency.

1.7 TECHNICAL TERMS

Data frame, excel , CSV (Comma Separated Values) , TSV (Tab-Separated Values)

1.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. 1. What is R programming, and why is it widely used in data analysis?

2. 2. How do you load data frames into R, and why are they important in data analysis?
3. 3. What are tab-separated value files, and how do you read data from them in R? Provide an example.
4. 4. How do you read data from Excel files in R? Which package and function do you use for this task?
5. 5. How do you check the structure and dimensions of a data frame in R after loading it from an external file?

Short Questions:

1. What are some common challenges or considerations when loading data into R from different files?
2. Explain the process of reading data from .CSV files in R. What function do you use, and what are its parameters?
3. Can you explain the difference between 'read.table()' and 'read.csv()' functions in R?
4. Can you demonstrate how to load multiple sheets from an Excel file into separate data frames in R?

1.9SUGGESTED READINGS

1. SeemaAcharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team
3. Crawley, M. J. (2012). The R book. John Wiley & Sons.
4. Albert, J. & Rizzo, M. (2012). R by Example. Springer
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R. Cambridge.

AUTHOR: **Dr. B. Reddaiah**

LESSON- 13

READING AND GETTING DATA INTO R EXTERNAL DATA

OBJECTIVES:

After going through this lesson, you will be able to

- Gain a comprehensive understanding of different data formats, including XML, JSON, and database formats.
- Acquire skills in retrieving data from various sources, including web APIs, online databases, and local files such as XML and JSON files
- Develop proficiency in parsing and processing different data formats, such as XML and JSON.
- Learn how to use R packages such as RMySQL or RSQLite to access and manipulate data stored in relational databases.

STRUCTURE OF THE LESSION:

13.1 XML FILES

XML (Extensible Markup Language) is a widely used markup language designed to store and transport structured data in a human-readable and machine-readable format. It provides a flexible and standardized way to represent hierarchical data, making it suitable for a wide range of applications, including data interchange, configuration files, web services, and document markup. XML documents consist of elements, attributes, and text content organized in a hierarchical tree-like structure. Each element can have nested child elements, and attributes provide additional metadata or properties associated with the elements. XML documents are typically defined by XML schemas or Document Type Definitions (DTDs), which specify the structure and constraints of the data being represented, ensuring interoperability and data integrity.

One of the key advantages of XML is its platform-independent and language-neutral nature, which allows data to be exchanged and processed across different systems and programming languages seamlessly. XML documents can be easily parsed and manipulated using a variety of programming languages and tools, making them a versatile choice for data representation

and exchange in distributed computing environments. Furthermore, XML supports extensibility and customization through user-defined schemas and namespaces, enabling developers to define their own data structures and vocabularies tailored to specific use cases or domains. This flexibility and extensibility make XML a popular choice for implementing data-centric applications and standards in various industries, including web development, enterprise integration, and scientific data exchange.

We can work with the XML files using the XML package provided by R. The package has to explicitly installed by the user

The following command is used to install this package:

```
install.packages("XML")
```

13.1.1 Installing the r.xml Package

To work with XML files in R, we need to install the r.xml package. This package provides essential functions for reading, parsing, and manipulating XML data.

1. Installing from CRAN

To install the r.xml package from CRAN, you can use the install. packages() function:

```
Install.packages("r.xml")
```

Make sure to execute this command in your R environment, and the package will be downloaded and installed automatically.

2. Loading the r.xml Package

After successful installation, load the r.xml package into your R session using the library() function:

```
library(r.xml)
```

By loading the package, you gain access to a variety of functions specifically designed for handling XML data efficiently.

13.1.2 Creating sample XML file

Before we dive into reading XML data in R using the r.xml package, let's create a sample XML file that we'll use for demonstration purposes. For this example, we'll continue with the bookstore theme from the previous section.

For this example, let's create an XML file representing a bookstore with two books. Each book will have attributes like category, and elements like title, author, year, and price.

1. Example XML File (books.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="Science Fiction">
<title lang="en">Station Eleven</title>
<author>Emily St. John Mandel</author>
<year> 2020</year>
<price>28.46</price>
</book>
<book category="Mystery">
<title lang="en">All the Dangerous Things. </title>
<author> Stacy Willingham</author>
<year> 2023 </year>
<price>35.71</price>
</book>
</bookstore>
```

In the above XML file, we have created a root element <books> that contains two child elements, <book>, representing individual books. Each <book> element has attributes like category, and several child elements, including <title>, <author>, <year>, and <price>, representing the book's details.

2. Explaining the XML Structure:

- **:** This is the XML declaration that specifies the XML version and encoding used in the document.
- **:** This is the root element of our XML file that contains all the book elements.
- **:** These are child elements under the <books> element, representing individual books. They have an attribute category to specify whether the book is fiction or non-fiction.

13.1.3 Reading XML Data in R

Now that we have our sample XML file, books.xml, created, let's dive into the process of reading and parsing XML data in R using the r.xml package. This step is crucial as it enables us to extract and work with the structured data stored within the XML file.

1. Understanding XML Tree Structure in r.xml

When an XML file is read and parsed using the r.xml package, it is transformed into a hierarchical tree-like structure known as the XML tree. Each element in the XML file becomes a node in the tree, and the relationships between elements are represented by parent-child connections.

To read and parse an XML file in R, we use the xmlTreeParse() function from the r.xml package.

Example - Reading and Parsing XML Data in R:

```
# Load the r.xml package
library(r.xml)

# Specify the file path of the XML data
xml_file <- "path/to/books.xml"  # Replace with the actual file
path

# Parse the XML data and create a tree structure
xml_tree <- xmlTreeParse(xml_file)

# Print the XML tree
print(xml_tree)
```

Output:

```
Titles: Station Eleven
Authors: Emily St. John Mandel
```

In the above example, we load the package into our R session and specify the file path to our sample XML file, books.xml. The xmlTreeParse() function is then used to read and parse the XML data from the file, creating an XML tree structure.

Navigating the XML Tree

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="Science Fiction">
<title lang="en">Station Eleven</title>
<author>Emily St. John Mandel</author>
<year> 2020</year>
<price>28.46</price>
</book>
<book category="Mystery">
<title lang="en"> ”All the Dangerous Things.”</title>
<author> Stacy Willingham</author>
<year> 2023 </year>
<price>35.71</price>
</book>
</bookstore>
```

To extract specific information from the XML tree, we need to navigate through its nodes. The r.xml package provides several functions for this purpose. One common function is getNodeSet(), which allows us to retrieve nodes based on their paths or criteria.

Example - Extracting Book Titles and Authors:

```
# Load the r.xml package
library(r.xml)

# Specify the file path of the XML data
xml_file <- "path/to/books.xml"  # Replace with the actual
file path

# Parse the XML data and create a tree structure
xml_tree <- xmlTreeParse(xml_file)

# Get all the <book> elements from the XML
book_elements <- getNodeSet(xml_tree, "//book")

# Extract book titles and authors
titles <- apply(book_elements, function(book)
xmlValue(xmlChildren(book)$title))
authors <- apply(book_elements, function(book)
xmlValue(xmlChildren(book)$author))

# Print the book titles and authors
cat("Titles:", titles, "\n")
cat("Authors:", authors, "\n")
```

Output:

In the above example, we use the getNodeSet() function to retrieve all the <book> elements from the XML tree. We then use sapply() to extract the titles and authors from each <book> element using xmlValue() and xmlChildren() functions.

Accessing Element Attributes

In addition to extracting element values, we can also access element attributes using the xmlGetAttr() function.

Example - Extracting Book Categories:

```
# Load the r.xml package
library(r.xml)

# Specify the file path of the XML data
xml_file <- "path/to/books.xml"  # Replace with the actual file
path

# Parse the XML data and create a tree structure
xml_tree <- xmlTreeParse(xml_file)

# Get all the <book> elements from the XML
book_elements <- getNodeSet(xml_tree, "//book")

# Extract book categories
categories <- sapply(book_elements, function(book)
xmlGetAttr(book, "category"))

# Print the book categories
cat("Categories:", categories, "\n")
```

Output:

```
Categories: Fiction Non-Fiction
```

In this example, we use xmlGetAttr() to access the category attribute of each <book> element, and sapply() to extract the category values from all the <book> elements.

13.1.4 XML to Data Frame Conversion

One of the most common tasks when working with XML data in R is converting it into a more familiar and tabular format, such as a data frame. Data frames are widely used in R for data manipulation and analysis, making it essential to learn how to convert XML data into this format. In this section, we will explore how to convert XML data into a data frame using the r.xml package.

1. Understanding Data Frame

A data frame is a two-dimensional tabular data structure in R, where rows represent observations and columns represent variables. Each column in a data frame can hold a different type of data, making it a versatile and powerful data structure for handling various types of data.

2. Converting XML to Data Frame

To convert XML data into a data frame, we first need to extract the relevant information from the XML elements and attributes. Once we have the data extracted, we can use the data.frame() function in R to create a data frame.

Example - Converting XML to Data Frame:

Let's consider the books.xml file we created earlier, which contains information about books in a books. We will convert this XML data into a data frame representing the book details.

```
# Load the r.xml package
library(r.xml)

# Specify the file path of the XML data
xml_file <- "path/to/books.xml"  # Replace with the actual file path

# Parse the XML data and create a tree structure
xml_tree <- xmlTreeParse(xml_file)

# Get all the <book> elements from the XML
book_elements <- getNodeSet(xml_tree, "//book")

# Initialize empty lists to store book details
titles <- vector("character", length = length(book_elements))
authors <- vector("character", length = length(book_elements))
years <- vector("integer", length = length(book_elements))
prices <- vector("numeric", length = length(book_elements))
categories <- vector("character", length = length(book_elements))

# Extract book details and attributes
for (i in seq_along(book_elements)) {
  book <- book_elements[[i]]
  titles[i] <- xmlValue(xmlFindNode(book, "//title"))
  authors[i] <- xmlValue(xmlFindNode(book, "//author"))
  years[i] <- as.integer(xmlValue(xmlFindNode(book, "//year")))
  prices[i] <- as.numeric(xmlValue(xmlFindNode(book, "//price")))
  categories[i] <- xmlGetAttr(book, "category")
}

# Create the data frame
book_df <- data.frame(
  Title = titles,
  Author = authors,
  Year = years,
  Price = prices,
  Category = categories
)

# Print the data frame
print(book_df)
```

```
Title  Author Year Price Category
1Station Eleven  Emily St. John Mandel  202028.46   Science Fiction
2 All the Dangerous Things. Stacy Willingham 2023 35.71   Mystery
```

In the above example, we first load the r.xml package and parse the XML data from the books.xml file. We then extract the book details and attributes (title, author, year, price, and category) from the <book> elements using the xmlValue() and xmlGetAttr() functions.

Next, we initialize empty lists to store the extracted data, and using a loop, we populate the lists with the respective values from each book element. Finally, we create a data frame named book_df using the data. frame() function, incorporating the extracted data and attributes as columns.

13.2 WEB DATA

Web data refers to any data that is sourced from the World Wide Web or internet. It encompasses a wide range of information that is publicly available on websites, web pages, social media platforms, online forums, and other online sources. Web data can include text, images, videos, structured data, metadata, user-generated content, and more. It is generated and consumed by individuals, organizations, and automated systems for various purposes, such as research, analysis, decision-making, and business intelligence.

Web data is often unstructured or semi-structured, meaning that it does not adhere to a strict schema or format, making it challenging to process and analyze using traditional methods. However, advancements in web scraping, data extraction, and natural language processing (NLP) techniques have made it possible to extract, parse, and derive insights from web data at scale. Web data is valuable for a wide range of applications, including market research, competitive analysis, sentiment analysis, trend detection, recommendation systems, and personalized content delivery. As the volume and complexity of web data continue to grow, leveraging this rich source of information effectively can provide organizations with valuable insights and a competitive edge in today's data-driven world.

In R, you can access web data using various packages and techniques, including web scraping, APIs, and direct file downloads. Here's a brief overview of each approach:

13.2.1 Web Scraping: Web scraping involves extracting data directly from HTML web pages. You can use packages like `rvest`, `xml2`, or `httr` along with XPath or CSS selectors to scrape data from specific elements on web pages. For example, you can extract tables, text, or images from websites and convert them into R data structures like data frames.

Web scraping using rvest:
'rvest' is an R package that provides tools for web scraping. It allows you to extract data from HTML web pages using CSS selectors or XPath expressions.

1. Installation: If you haven't already installed the 'rvest' package, you can do so from CRAN using the following command:

```
install.packages("rvest")
```

2. Loading the Package: After installation, load the 'rvest' package into your R session using the 'library()' function:

```
library(rvest)
```

3. Scraping Data: You can use the 'read_html()' function to retrieve the HTML content of a web page and then use CSS selectors or XPath expressions with functions like 'html_nodes()' and 'html_text()' to extract specific elements or text from the HTML.

Read the HTML content of a web page

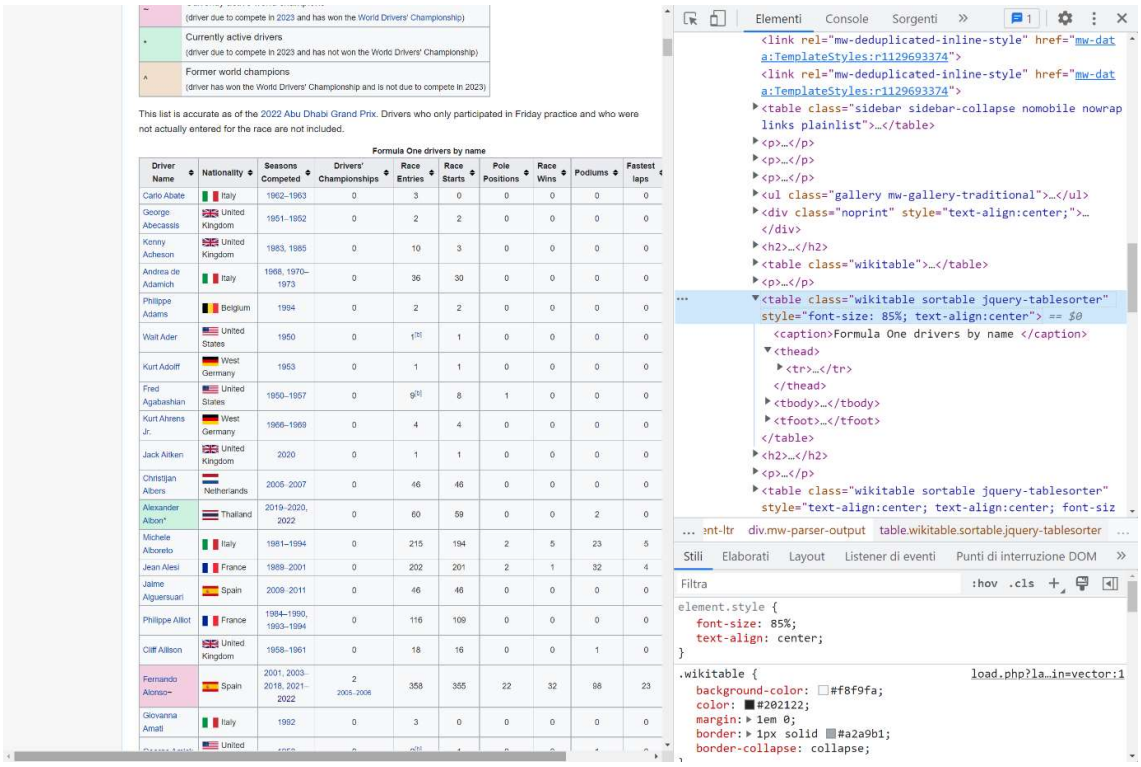
```
url <- "https://www.nytimes.com/"
page <- read_html(url)
page
```

Extract text from all <p> elements on the page

```
paragraphs <- page %>% html_nodes("p") %>% html_text()
```

Extract text from the first <h1> element on the page

```
heading <- page %>% html_nodes("h1") %>% html_text()
```



4. Further Processing: Once we've extracted the desired data, you can further process it using R's built-in functions or other packages. For example, you can convert the extracted data into a data frame for analysis or visualization.

Convert extracted data into a data frame

```
data_frame <- data.frame(Paragraphs = paragraphs, Heading = heading)
```

'rvest' provides a powerful and flexible way to scrape data from HTML web pages in R. However, it's important to be mindful of website terms of service and to scrape responsibly and ethically. Additionally, some websites may have rate limits or require authentication to access their content, so be sure to check the website's policies and documentation before scraping.

13.2.2 APIs (Application Programming Interfaces): Many websites and online platforms offer APIs that allow developers to access and retrieve data in a structured format. We can use packages like `httr` or specialized API client packages (e.g., `twitteR` for Twitter API) to interact with APIs and fetch data directly into R.

'httr' is an R package that provides tools for working with HTTP requests and responses. It allows you to interact with web APIs, download files from the web, and perform other HTTP-related tasks. Here's how you can use 'httr' in R:

1. Installation: If you haven't already installed the 'httr' package, you can do so from CRAN using the following command:

```
install.packages("httr")
```

2. Loading the Package: After installation, load the 'httr' package into your R session using the 'library()' function:

```
library(httr)
```

3. Making HTTP Requests: You can use functions like 'GET()', 'POST()', 'PUT()', and 'DELETE()' to make HTTP requests to web servers. These functions allow you to specify the URL, request headers, request body, and other parameters as needed.

Make a GET request to a web API

```
response <- GET("https://api.example.com/data")
```

Make a POST request with JSON data

response <- POST("https://api.example.com/submit", body = list(name = "John", age = 30), encode = "json")

4. Handling Responses: After making a request, you can inspect the response object to access the response status code, headers, content, and other information. The 'content()' function allows you to extract the content of the response in various formats, such as text, JSON, or parsed XML.

Check the response status code

```
status_code(response)
```

Extract JSON content from the response

```
json_content <- content(response, "parsed")
```

5. Downloading Files: You can use the 'GET()' function to download files from the web. The 'write_disk()' function allows you to save the downloaded content to a file on your local filesystem.

Download a file from a URL

```
response <- GET("https://example.com/data.csv")
```

Save the downloaded content to a file

```
write_disk(content(response), path = "data.csv")
```

'httr' provides a flexible and powerful interface for working with HTTP requests and responses in R. It is commonly used for interacting with web APIs, accessing web resources, and downloading files from the web.

5. Convert raw data to char format

To convert raw data in char format we need to use rawToChar() and pass variable_name$content in it just like we did in this example

```
# installing packages
install.packages("httr")

# importing packages
library(httr)

# GET() method will store the raw
# data in r variable
r < - GET("https://example.com")

# rawToChar() will convert raw data
# to char and store in response variable
response < - rawToChar(r$content)

# print response
print(response)
```

Output:


"<!doctype html><html lang=en-us prefix=\"og: http://ogp.me/ns#\"><meta charset=utf-8><meta name=viewport content=\"width=device-width,initial-scale=1,maximum-scale=1\"><meta property=\"og
m/js/platform.js></script><script async src=//cdnjs.cloudflare.com/ajax/libs/require.js/2.1.14/require.min.js></script><script async src=\"//translate.google.com/translate_a/element.js?cb=
{\r\n \"@type\" : \"Person\",\r\n \"name\" : \"Sandeep Jain\",\r\n \"url\" : \"https://in.linkedin.com/in/sandeep-jain-b3940815\"\r\n
var isAdminLoggedIn = 0;\r\n </script><meta property=\"og:locale\" content=\"en_US\"><meta property=\"og:type\" content=\"website\"><meta property=\"og:title\" c
ments/feed/><script>\n\t\t\twindow._wpemojiSettings = {\"baseUrl\":\"https:\\/\\/s.w.org\\/images\\/core\\/emoji\\/11\\/72x72\\/\",\"ext\":\".png\",\"svgUrl\":\"https:\\/\\/s.w.org\\/image
3,56128,56447]),!b);case\"emoji\":return b=d([55358,56760,9792,65039],[55358,56760,8203,9792,65039]),!b}return!1}function f(a){var c=b.createElement(\"script\");c.src=a,c.defer=c.type=\"te
mplete\"===b.readyState&&c.readyCallback())}),g=c.source||{},g.concatemoji?f(g.concatemoji):g.wpemoji&&g.twemoji&&(f(g.twemoji),f(g.wpemoji))}(window,document,window._wpemojiSettings);\n\
\/\\/util.geeksforgeeks.org\\/\",\"apiUrl\":\"https:\\/\\/api.geeksforgeeks.org\\/\"};\n/*]]> */\n</script><script async src=\"https://www.geeksforgeeks.org/wp-content/themes/iconic-one/
uploads/gfg_200X200.png><meta name=msapplication-TileImage content=\"https://www.geeksforgeeks.org/wp-content/uploads/gfg_200X200.png\"><style>@media(max-width:1043px) and (min-width:992px
_timer=setTimeout(function () {_vwo_code.finish() },settings_tolerance);var a=d.createElement('style'),b=hide_element?hide_element+'{opacity:0 !important;filter:alpha(opacity=0) !important
stom-background custom-background-white\"><script>\r\n // Dark mode script\r\n const gfgThemeList = Object.freeze({\r\n LIGHT: \"gfgThemeLight\",\r\n DA
let [mainImg, extractImage] = /(.*)\\.svg/g.exec(imageSrc)\r\n if (!mainImg.includes(\"%40dark\") && theme === gfgThemeList.DARK) {\r\n mainImg = `${ex
9.026\"><path d=\"M2380.7 6597.866a12.252 12.252.0 00-.261-1.5131-30.726-.027a12.545 12.545.0 01.908-3.443 12.337 12.337.0 012.739-4.044 12.151 12.151.0 014.018-2.581 12.634 12.634.0 0114.
19.545.0 004.226 6.184 19.111 19.111.0 006.261 4.07 20.241 20.241.0 007.671 1.46 20.572 20.572.0 007.981-1.46 17.282 17.282.0 006-4.123 15.717 15.717.0 003.445-5.686 20.726 20.726.0 001.1-
ne 1\" x1=\"1.046\" fill=\"#0f2b3c\"/></g><g id=\"Group_157\" data-name=\"Group 157\" transform=\"translate(56.695 0.004)\"><path id=\"Path_243\" data-name=\"Path 243\" d=\"M1352.9 615.759
049.0 001.879-.395 1.266 1.266.0 00.763-1.185 1.367 1.367.0 00-.654-1.2 6.492 6.492.0 00-2.016-.7631-1.662-.409a5.974 5.974.0 01-2.752-1.444 3.452 3.452.0 01-1.089-2.615 3.4 3.4.0 01.722-2
9 -613.146)\" fill=\"#0f2b3c\"/></g><g id=\"Group_161\" data-name=\"Group 161\" transform=\"translate(29.011 6.524)\"><path id=\"Path_247\" data-name=\"Path 247\" d=\"M1261.615 621.145c.01
id=\"Path_249\" data-name=\"Path 249\" d=\"M1430.636 598.566a2.5 2.5.0 011.8-.549 4.2 4.2.0 011.657.281V-2.752a7.24 7.24.0 00-2.3-.337 4.82 4.82.0 00-3.455 1.278 4.666 4.666.0 00-1.349 3.5
\"Group_166\" data-name=\"Group 166\" transform=\"translate(194.036 6.509)\"><path id=\"Path_252\" data-name=\"Path 252\" d=\"M1737.51 626.817a5.667 5.667.0 01-2.152-2.683l2.67-1.171a3.742
6.524)\"><path id=\"Path_253\" data-name=\"Path 253\" d=\"M1613.182 621.145c.017-.232.027-.466.025-.7a7.375 7.375.0 10-14.749.155c0 .236.017.47.04.7a7.254 7.254.0 00.673 2.382 7.375 7.375.
1h9.895Q1554.01 604.7 1554.01 605.147z\" transform=\"translate(-1532.736 -594.51)\" fill=\"#0f2b3c\"/></g></svg> <span
ded=false data-expandable=false>DSA Topic-wise<li class=mega-dropdown__list-item data-child=true aria-expanded=false
able=false>Worst, Average and Best Cases<li class=mega-dropdown__list-i
is-of-algorithms-set-4-analysis-of-loops/?ref=ghm\" target=_self>Analysis of Loops<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\
ga-dropdown__list-item data-child=true aria-expanded=false data-expandable=false>Polyno
lgorithms/?ref=ghm\" target=_self>Graph Algorithms<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksforgeeks.org/a
ized-algorithms/?ref=ghm\" target=_self>Randomized Algorithms<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksfor
bound-algorithm/?ref=ghm\" target=_self>Branch and Bound<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksforgeeks
?ref=ghm\" target=_self>Stack<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksforgeeks.org/queue-data-structure/?
hing<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/?ref=ghm\"
ropdown__list-item data-parent=true aria-expanded=true data-expandable=true>Interview Corner<i class=\"gfg-icon gfg-icon_arrow-right\"></i><ul class=mega-dropdown><li class=me
<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksforgeeks.org/experienced-interview-experiences-company-wise/?ref
w.geeksforgeeks.org/system-design-tutorial/ target=_self>System Design Tutorial<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"ht
/a><li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false>
anded=false data-expandable=false>PHP<li class=mega-dropdown__list-item data-child=true aria-expanded=fals
utorials/?ref=ghm\" target=_self>CSS<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://www.geeksforgeeks.org/kotlin-programm
i class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false>Comput
eeks.org/digital-electronics-logic-design-tutorials/#blg/?ref=ghm\" target=_self>Digital Logic<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=fal
eeks.org/gate-corner-2-gq/?ref=ghm\" target=_self>GATE CS Solved Papers<li class=mega-dropdown__list-item data-child=true aria-expanded=false data-expandable=false><a href=\"https://ww

13.3 JSON FILES

A JSON (JavaScript Object Notation) file is a text-based data format commonly used for storing and exchanging structured data between systems and applications. JSON files are human-readable and easy to understand, making them popular for data interchange over the internet. JSON is based on a subset of the JavaScript programming language, but it is language-independent, meaning it can be parsed and generated by virtually any programming language.

JSON files consist of key-value pairs organized into objects and arrays. An object is an unordered collection of key-value pairs enclosed in curly braces `{}`, where each key is a string and each value can be a string, number, boolean, object, array, or null. An array is an ordered collection of values enclosed in square brackets `[]`, where each value can be a string, number, boolean, object, array, or null. JSON files can also contain nested objects and arrays, allowing for complex data structures to be represented.

To deal with JSON files in R language, the rjson package is required. You can install the rjson package through the following command:

```
install.packages("rjson")
```

The rjson library helps perform multiple tasks, like:

- Reading JSON files.
- Converting JSON into a dataframe or table.

Reading JSON files

To read a JSON file, you can use the from JSON() method of the rjson library.

we have two files named main.r and data.json. main.r contains code to read the data.json file using the fromJSON() method.

main.r

```
# Load the package required to read JSON files.
install.packages("rjson") # Optional
library("rjson")
# Give the input file name to the function.
myData <- fromJSON(file="data.json")
# Print the result.
print(myData)
```

data.json

```
{
   "ID":["1","2","3","4","5","6","7","8" ],

"Name":["James","Robert","Eric","Ryan","Daniel","Mark","Paul","Gary"
],
   "Age":["32","41","21","29","43","48","33","42" ],
   "JoiningDate":[
"1/10/2013","9/22/2014","11/16/2015","5/13/2016","4/25/2017","4/11/2
018","8/20/2019","9/19/2020"],
   "Dept":[
"Business","Operations","HR","IT","Accounts","Business","Operations"
,"Accounts"]
}
```

Output

```
> library("rjson")
> myData <- fromJSON(file="data.json")
> print(myData)
$ID
[1] "1" "2" "3" "4" "5" "6" "7" "8"

$Name
[1] "James"  "Robert" "Eric"   "Ryan"   "Daniel" "Mark"   "Paul"
[8] "Gary"

$Age
[1] "32" "41" "21" "29" "43" "48" "33" "42"

$JoiningDate
[1] "1/10/2013"  "9/22/2014"  "11/16/2015" "5/13/2016"  "4/25/2017"
[6] "4/11/2018"  "8/20/2019"  "9/19/2020"

$Dept
[1] "Business"   "Operations" "HR"         "IT"         "Accounts"
[6] "Business"   "Operations" "Accounts"

> |
```

Convert JSON into a data frame

To convert a JSON file to a data frame, you can use the as.data.frame() method.

We simply use the fromJSON() function to read data from the data.json file and pass loaded data to the as.data.frame() method to convert into a data frame.

we have two files named main.r and data.json. main.r contains code to read the data.json file to the myData variable. as.data.frame(myData) helps to convert raw JSON data into a data frame.

main.r

```
# Load the package
# required to read JSON files.
library("rjson")
# Passing argument files
myData <- fromJSON(file="data.json")
# Convert JSON file to dataframe.
json_data_frame <- as.data.frame(myData)
print(json_data_frame)
```

data.json

```
{
   "ID":["1","2","3","4","5","6","7","8" ],

"Name":["Scott","Frank","Eric","Frank","Daniel","Mark","Paul","Gary"
],
   "Age":["32","41","21","29","43","48","33","42" ]
}
```

Output:

```
> library("rjson")
> myData <- fromJSON(file="data.json")
> json_data_frame <- as.data.frame(myData)
> print(json_data_frame)
  ID   Name Age JoiningDate       Dept
1  1  James  32   1/10/2013   Business
2  2 Robert  41   9/22/2014 Operations
3  3   Eric  21  11/16/2015         HR
4  4   Ryan  29   5/13/2016         IT
5  5 Daniel  43   4/25/2017   Accounts
6  6   Mark  48   4/11/2018   Business
7  7   Paul  33   8/20/2019 Operations
8  8   Gary  42   9/19/2020   Accounts
```

13.4 DATABASES

Database systems that use relationships store data in a standardized manner. Therefore, in order to execute statistical computing, you will need to write very sophisticated and complex SQL queries. On the other hand, R can easily establish a connection and obtain records as a data frame from a variety of relational databases, including MySQL, Oracle, and SQL Server. When data enters the R environment, it becomes a typical R data set that can be worked with and examined using all of the advanced functions and packages available in R.

Connecting R to MySQL

`RMySQL` is an R package that provides an interface to MySQL databases. It allows you to connect to a MySQL database from within R, execute SQL queries, and retrieve data into R data structures for further analysis and manipulation. Here's a step-by-step guide on how to use `RMySQL`:

1. Installation: If you haven't already installed the `RMySQL` package, you can do so from CRAN using the following command:

```
install.packages("RMySQL")
```

2. Loading the Package: After installation, load the `RMySQL` package into your R session using the `library()` function:

```
library(RMySQL)
```

3. Establishing a Connection: Use the `dbConnect()` function to establish a connection to your MySQL database. You'll need to provide the necessary connection details, such as the host, port, username, password, and database name.

Replace 'host', 'user', 'password', and 'dbname' with your actual connection details

```
con <- dbConnect(MySQL(),
                 host = 'localhost',
                 user = 'root',
                 password = 'password',
                 dbname = 'mydatabase')
```

4. Executing Queries: Once the connection is established, you can execute SQL queries using the `dbSendQuery()` function. This function sends the query to the database server but does not fetch the results immediately.

```
# Execute an SQL query
query <- "SELECT * FROM mytable"
result <- dbSendQuery(con, query)
```

5. Fetching Results: To retrieve the results of a query, you can use the `dbFetch()` function. This function fetches the results from the database server and returns them as an R data frame.

Fetch the results of the query into a data frame

```
data <- dbFetch(result)
```

6. Closing the Connection: After you've finished working with the database, it's a good practice to close the connection using the `dbDisconnect()` function.

Close the database connection

```
dbDisconnect(con)
```

By following these steps, you can connect to a MySQL database from R, execute SQL queries, and work with the data using `RMySQL`. This package is useful for accessing and analyzing data stored in MySQL databases directly from R.

1.5 SUMMARY

The chapter covers a comprehensive range of topics related to data acquisition and management, including XML files, web data, JSON files, and databases. It starts by introducing XML files as a structured data format commonly used for representing hierarchical data. Techniques for reading and processing XML files in R, such as using the XML package, are discussed. The chapter then delves into web data access, highlighting methods for web scraping, interacting with web APIs, and downloading files from the web using packages like rvest and httr. Additionally, it explores JSON files as a popular data interchange format and demonstrates how to read and manipulate JSON data in R using the jsonlite package. Furthermore, the chapter covers accessing and querying databases directly from R, with a focus on relational databases like MySQL and packages such as RMySQL for establishing connections and executing SQL queries. Overall, the chapter equips readers with the essential skills and techniques for acquiring, processing, and managing various types of data sources in R for analysis and visualization purposes.

1.6 TECHNICAL TERMS

XML, JSON, web data, SQL Queries

1.7 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What are some common applications of XML in web development and data interchange?
2. How do you access and extract data from XML files using R, and what are some common challenges in working with XML data?
3. What are APIs (Application Programming Interfaces), and how are they used to access web data?
4. How do you read and parse JSON files in R, and what are some advantages of using the jsonlite package for this task?
5. How do you connect to and query a database from R, and what are some common functions and packages used for database interaction?

Short Questions:

1. What is XML, and how is it different from other data formats like JSON and CSV?
2. Can you explain the structure of an XML file and how hierarchical data is represented within it?

3. What is web data, and how is it different from other types of data?
4. Can you describe the process of web scraping and its applications in data collection and analysis?
5. How do you make HTTP requests in R to interact with web APIs, and what are some common HTTP methods used for this purpose?
6. What is JSON, and why is it commonly used for representing structured data on the web?

1.8 SUGGESTED READINGS

1. Seema Acharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team
3. Crawley, M. J. (2012). The R book. John Wiley & Sons.
4. Albert, J. & Rizzo, M. (2012). R by Example. Springer
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R. Cambridge.

AUTHOR: **Dr. B. Reddaiah**

LESSON- 14

WORKING WITH R CHARTS AND GRAPHS

OBJECTIVES:

After going through this lesson, you will be able to

- Comprehend the importance of data visualization in EDA
- Gain familiarity with common plot types
- How to create histograms in R to visualize the distribution of a single variable.
- Learn to generate bar charts, line graphs, scater plots and pie charts

STRUCTURE OF THE LESSION:

14.1 HISTOGRAMS

In R, histograms are widely used for visualizing the distribution of data. They provide a quick and effective way to explore the shape and spread of a dataset.

Bins: The range of data values is divided into intervals called "bins" or "buckets." These bins are usually of equal width, although they can vary depending on the context.

Frequency: For each bin, the number of data points that fall within that bin's range is counted. This count is called the frequency.

Bars: The frequencies are then represented as bars, with the height of each bar corresponding to the frequency of data points in that bin.

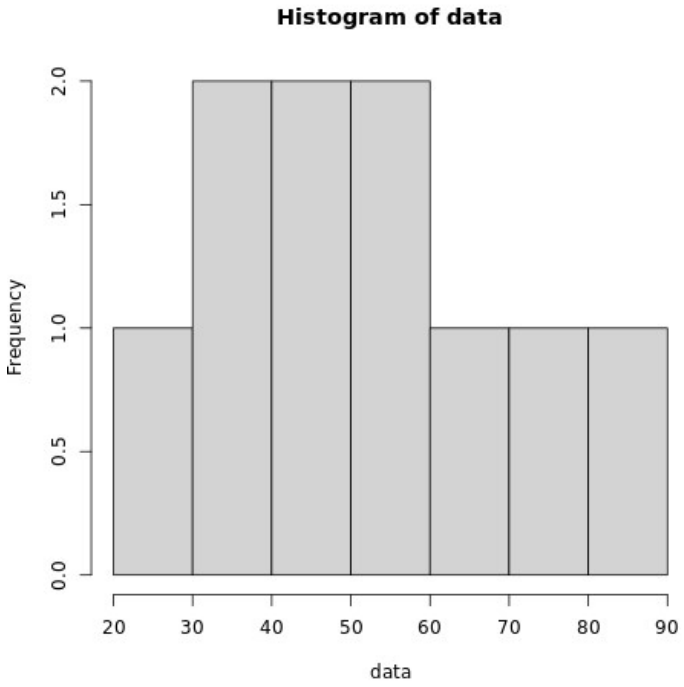
1. Creating a Histogram: You can create a histogram using the 'hist()' function. This function takes a vector of numeric values as input and produces a histogram plot.

Create a vector of numeric values (e.g., a dataset)

```
data<- c(23, 45, 67, 34, 56, 78, 89, 34, 56, 45)
```

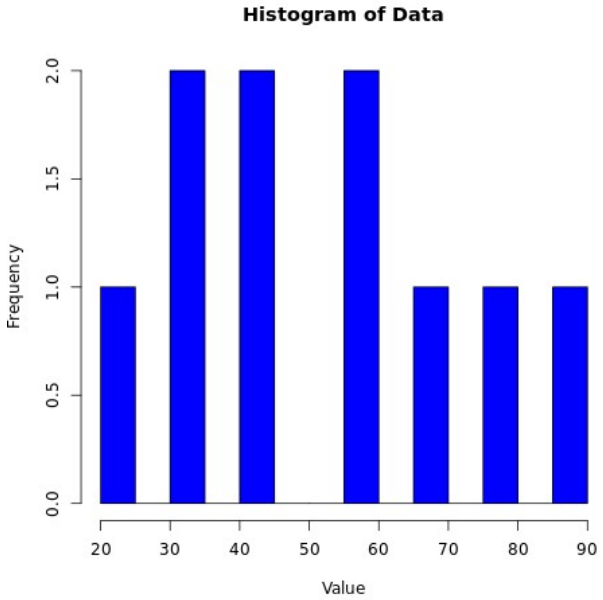
Create a histogram of the data

```
hist(data)
```



2. Customizing Histograms: You can customize various aspects of the histogram, such as the number of bins, axis labels, titles, colors, and more.

```
# Customizing the histogram
hist(data,
breaks = 10,            # Number of bins
col = "skyblue",        # Color of bars
xlab = "Value",         # Label for x-axis
ylab = "Frequency",     # Label for y-axis
main = "Histogram of Data")  # Title
```



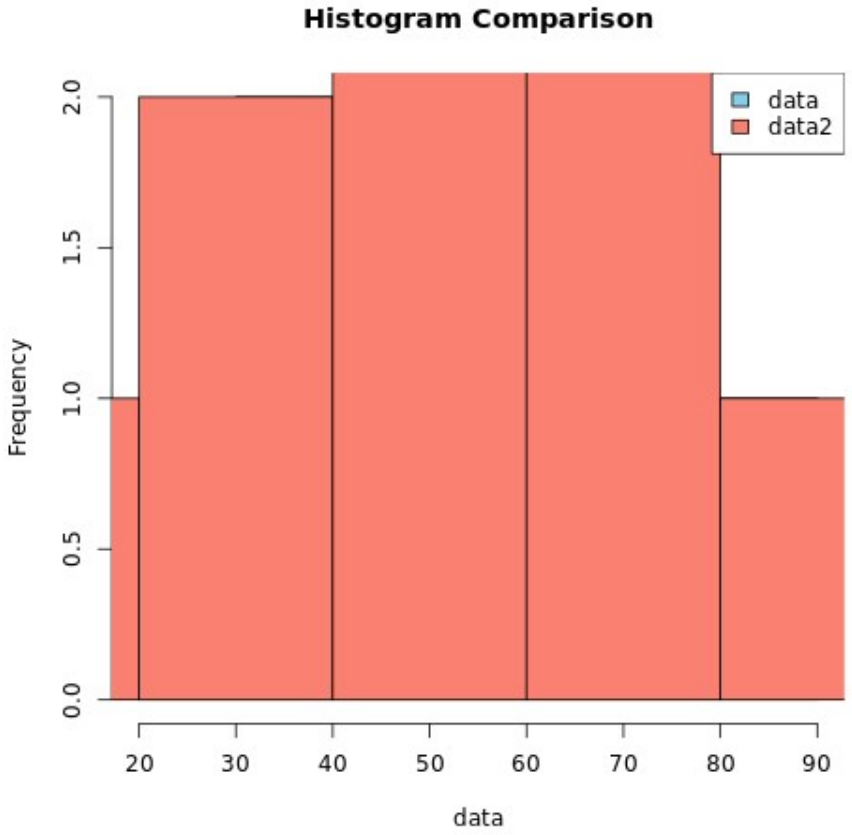
3. Overlaying Histograms: You can overlay multiple histograms on the same plot to compare different datasets or groups.

Create another dataset

```
data2 <- c(12, 45, 67, 78, 34, 56, 90, 23, 45, 67)
```

Overlay two histograms

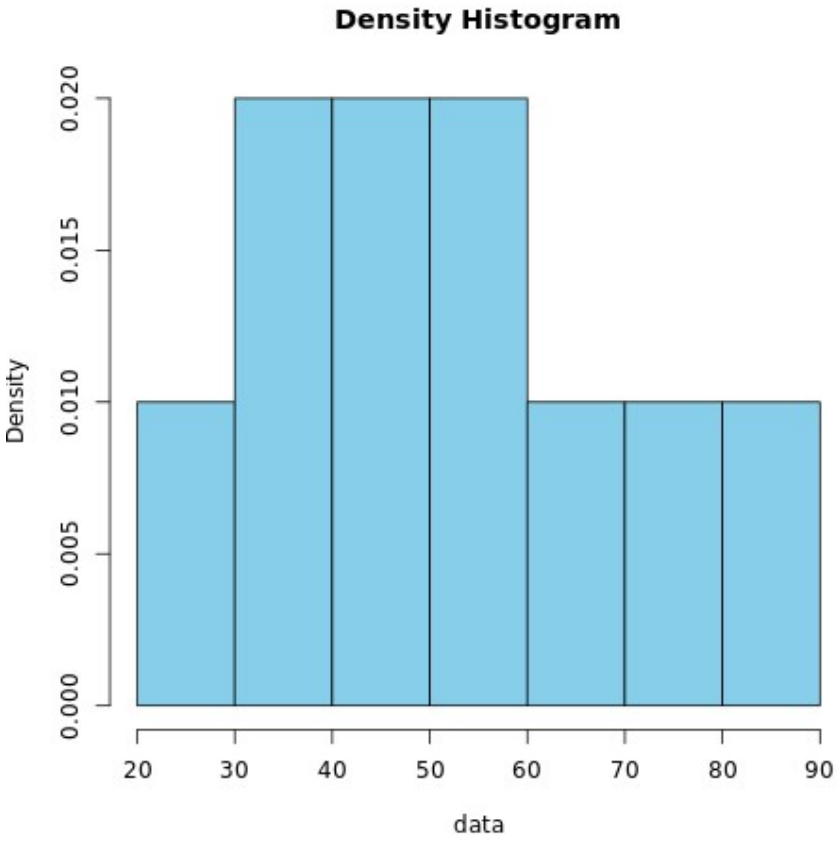
```
hist(data, col = "skyblue", main = "Histogram Comparison")
hist(data2, col = "salmon", add = TRUE)
legend("topright", legend = c("data", "data2"), fill = c("skyblue", "salmon"))
```



4. Density Histograms: Instead of plotting frequencies, you can plot probability densities using the 'freq = FALSE' argument.

Density histogram

```
hist(data, freq = FALSE, col = "skyblue", main = "Density Histogram")
```



Histograms in R are incredibly versatile and provide valuable insights into the distribution of data, making them a fundamental tool in data analysis and visualization workflows.

13.2 BAR CHARTS

Creating bar charts in R is a common task and can be accomplished using various packages, but the base R 'barplot()' function is a good place to start. Here's a basic example:

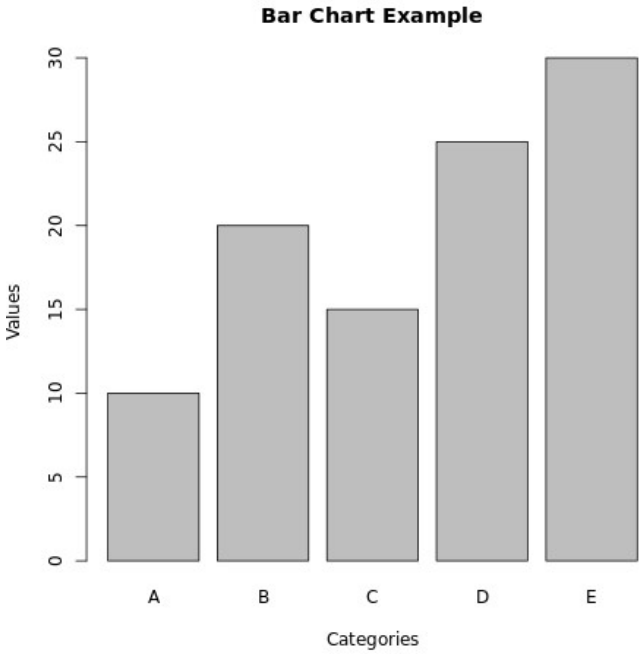
Sample data

```
data<- c(10, 20, 15, 25, 30)
names<- c("A", "B", "C", "D", "E")
```

Create a bar plot

```
barplot(data, names.arg = names, xlab = "Categories", ylab = "Values", main = "Bar Chart Example")
```

This code creates a simple bar chart with the values specified in the 'data' vector and the corresponding labels provided in the 'names' vector. The 'names.arg' argument specifies the labels for each bar, while 'xlab', 'ylab', and 'main' are used to label the axes and title of the plot.



If you want to customize your bar chart further, you might want to explore additional parameters of the 'barplot()' function or use other packages like 'ggplot2' for more advanced and customizable plots.

13.3 LINE GRAPHS,

Creating line graphs in R is commonly done using the base R 'plot()' function or the more powerful and flexible 'ggplot2' package. Here's an example of both approaches:

Using base R 'plot()' function:
Sample data

```
x <- 1:10
y <- x^2
```

Create a line plot

```
plot(x, y, type = "l", xlab = "X-axis", ylab = "Y-axis", main = "Line Graph Example")
```

In this example, 'x' and 'y' are vectors representing the x and y coordinates of the data points. The 'type = "l"' argument specifies that we want to plot lines. Other options for the 'type' argument include '"p"' for points, '"b"' for both points and lines, and more.

Using 'ggplot2' package:
Load the ggplot2 package

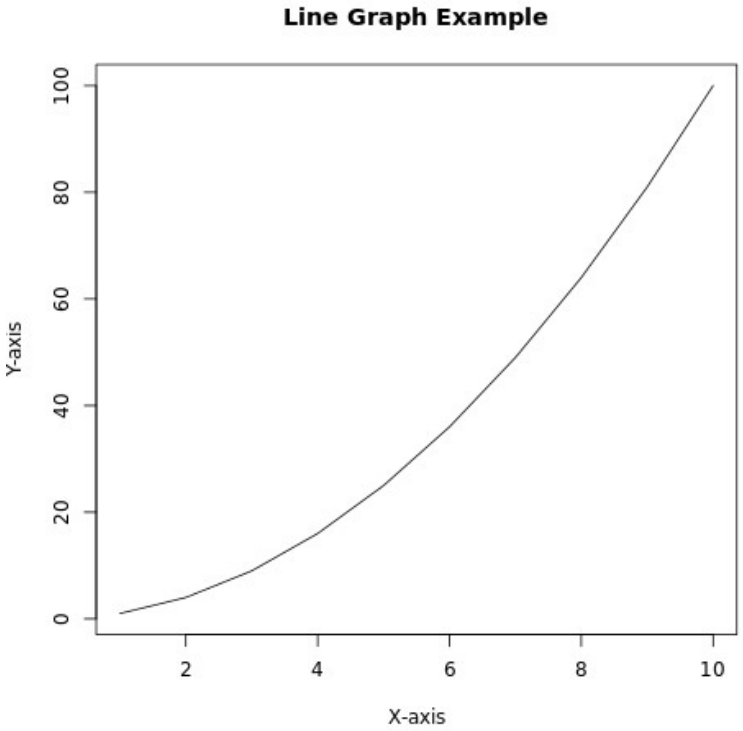
```
library(ggplot2)
```

Create a data frame

```
data<- data.frame(x = 1:10, y = (1:10)^2)
```

Create a line plot using ggplot2

```
ggplot(data, aes(x = x, y = y)) +
geom_line() +
labs(x = "X-axis", y = "Y-axis", title = "Line Graph Example")
```



In this 'ggplot2' example, we first create a data frame 'data' containing our x and y values. Then, using 'ggplot()', we specify the data and aesthetics (mapping our variables to x and y axes). 'geom_line()' adds the line layer to the plot, and 'labs()' is used to label the axes and title.

Both methods produce line graphs, but 'ggplot2' offers more customization options and is generally more versatile for creating complex visualizations.

13.4 SCATTER PLOTS

Creating scatter plots in R can also be done using the base R 'plot()' function or the 'ggplot2' package. Here's an example of both:

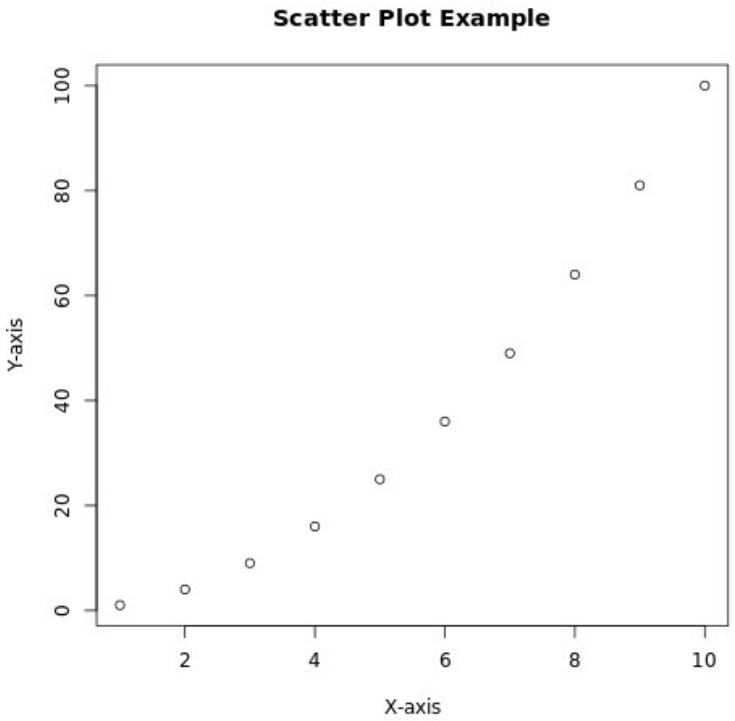
Using base R 'plot()' function:

Sample data

```
x <- 1:10
y <- x^2
```

Create a scatter plot

```
plot(x, y, xlab = "X-axis", ylab = "Y-axis", main = "Scatter Plot
Example")
```



In this example, 'x' and 'y' are vectors representing the x and y coordinates of the data points. The 'plot()' function creates a scatter plot by default when given two vectors of data.

Using 'ggplot2' package:

Load the ggplot2 package

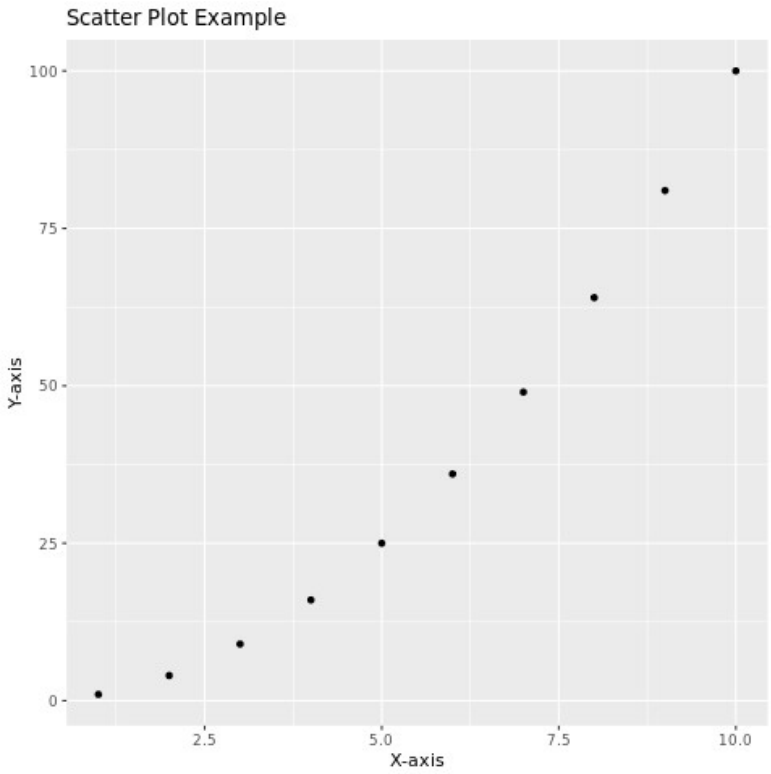
```
library(ggplot2)
```

Create a data frame

```
data<- data.frame(x = 1:10, y = (1:10)^2)
```

Create a scatter plot using ggplot2

```
ggplot(data, aes(x = x, y = y)) +
geom_point() +
labs(x = "X-axis", y = "Y-axis", title = "Scatter Plot Example")
```



In this 'ggplot2' example, we first create a data frame 'data' containing our x and y values. Then, using 'ggplot()', we specify the data and aesthetics (mapping our variables to x and y axes). 'geom_point()' adds the points layer to the plot, creating a scatter plot, and 'labs()' is used to label the axes and title.

Both methods produce scatter plots, but 'ggplot2' offers more customization options and is generally more versatile for creating complex visualizations.

13.5 PIE CHARTS

Creating pie charts in R is straightforward, and you can use the base R 'pie()' function to generate them. Here's an example:

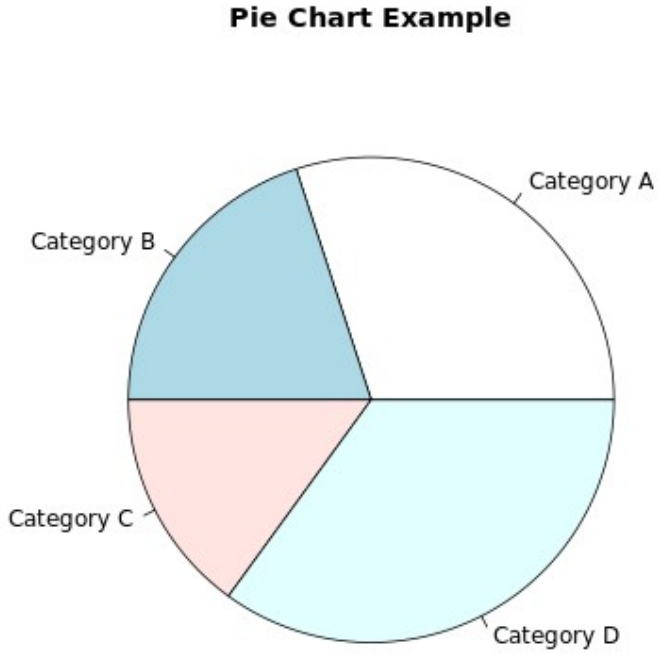
Sample data

```
data<- c(30, 20, 15, 35)
labels<- c("Category A", "Category B", "Category C", "Category D")
```

Create a pie chart

```
pie(data, labels = labels, main = "Pie Chart Example")
```

In this example, 'data' contains the numeric values for each slice of the pie chart, and 'labels' contains the corresponding labels for each slice. The 'pie()' function creates the pie chart, and you can specify additional parameters such as 'main' for the title of the chart.



While pie charts are commonly used, it's essential to note that they have limitations, especially when it comes to comparing the sizes of different categories accurately. In many cases, bar charts or other types of visualizations might be more suitable.

13.6 SUMMARY

The chapter delves into various visualization techniques, encompassing histograms, bar charts, line graphs, scatter plots, and pie charts. Each visualization method serves distinct purposes, enabling the representation and analysis of different types of data. Histograms provide insights into the distribution of continuous data by displaying frequency distributions across defined intervals. Bar charts are effective for comparing categorical data by depicting the frequencies or proportions of different categories using bars of varying heights. Line graphs elucidate trends and relationships in continuous data over time or another continuous variable. Scatter plots reveal patterns and relationships between two continuous variables, facilitating the identification of correlations or trends. Lastly, pie charts offer a simple yet intuitive way to represent the composition or proportions of categorical data, making them suitable for displaying parts of a whole. Through these visualization techniques, analysts and researchers can explore and communicate key insights derived from their data effectively.

13.7 TECHNICAL TERMS

Histograms, Bar Charts, Line Graphs,, Scatter plots, Pie Charts

13.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the purpose of histograms in data visualization and provide an example of when they are useful.
2. Describe the key components of a histogram and how they contribute to understanding the distribution of data.

3. Compare and contrast histograms with bar charts. When would you choose to use one over the other?
4. Provide an example of a dataset where a histogram would be the most appropriate visualization method.
5. What are the main steps involved in creating a histogram in R? Provide a code example.
6. Explain the purpose of bar charts and provide real-world scenarios where they are commonly used.

Short Questions:

1. Discuss the advantages and disadvantages of using bar charts in data visualization.
2. Provide a code example in R for creating a bar chart using both base R and the ggplot2 package.
3. Describe the characteristics of line graphs and when they are particularly useful for visualizing data.
4. Provide an example of a dataset where a line graph would be an appropriate visualization method.
5. Discuss the advantages of using scatter plots in data analysis and visualization.
6. Describe the main components of a pie chart and when it is appropriate to use one.

13.9SUGGESTED READINGS

1. SeemaAcharya , Data Analytics using R, McGraw Hill Education (India) pvt. ltd.
2. An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics. W. N. Venables, D.M. Smith and the R Development Core Team
3. Crawley, M. J. (2012). The R book. John Wiley & Sons.
4. Albert, J. & Rizzo, M. (2012). R by Example. Springer
5. Braun, W. j. & Murdoch, D. J. (2007). A First Course in Statistical Programming with R. Cambridge.

AUTHOR: **Mrs. A. Sarvani**