**SRI RACHAPUDY NAGABHUSHANAM DEGREE & PG COLLEGE, BADVEL**

**B.COM(CA) FINAL YEAR – VI SEM**

**BIG DATA**

**MATERIAL**

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

Data, classification Of Digital Data--structured, unstructured, semi-structured data, characteristics of 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, 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.

**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

**Unit – 1:**

**Introduction to Big data**

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**UNIT-I**

**Introduction to big data**

**What is Data?**

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

**What is Big Data?**

Big Data is also **data** but with a **huge size**. Big Data is a term used to describe a collection of data that is huge in size and yet growing exponentially with time. In short such data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

**Big Data** is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also a data but with huge size.

## Types Of Big Data

Big Data could be found in three forms:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

### ****Structured****

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Structured data in the form of rows and columns.

**Examples Of Structured Data**

An 'Employee' table in a database is an example of Structured Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee\_ID** | **Employee\_Name** | **Gender** | **Department** | **Salary\_In\_lacs** |
| 2365 | Subbaiah | Male | Finance | 650000 |
| 3398 | Visista | Female | Admin | 650000 |
| 7465 | Rajendra | Male | Admin | 500000 |
| 7500 | Venkata Subbaiah | Male | Finance | 500000 |
| 7699 | Haritha | Female | Finance | 550000 |

## ****Unstructured****

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. Unstructured data have no clear format in storage.

A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc.

**Semi-structured**

Semi-structured data can contain both the forms of data. It is very difficult to categorize this type of data.Some times they look structured or sometimes unstructured.

Examples Of Semi-structured Data is face book, twitter and YouTube.

## Characteristics Of Big Data?

**(i) Volume –** The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data.

**(ii) Variety –** The next aspect of Big Data is its **variety**.

It refers to the format of data which is classified as: Structured, unstructured and semi-structured data.

**(iii) Velocity –** The term **'velocity'** refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data.

**Challenges of Big Data?**

**Storage**

With vast amounts of data generated daily, the greatest challenge is storage (especially when the data is in different formats) within legacy systems. Unstructured data cannot be stored in traditional databases.

### Processing

Processing big data refers to the reading, transforming, extraction, and formatting of useful information from raw information. The input and output of information in unified formats continue to present difficulties.

### Security

Security is a big concern for organizations. Non-encrypted information is at risk of theft or damage by cyber-criminals. Therefore, data security professionals must balance access to data against maintaining strict security protocols.

### Finding and Fixing Data Quality Issues

Many of you are probably dealing with challenges related to poor data quality, but solutions are available. The following are four approaches to fixing data problems:

* Correct information in the original database.
* Repairing the original data source is necessary to resolve any data inaccuracies.
* You must use highly accurate methods of determining who someone is.

### Scaling Big Data Systems

Database sharding, memory caching, moving to the cloud and separating read-only and write-active databases are all effective scaling methods. While each one of those approaches is fantastic on its own, combining them will lead you to the next level.

### Evaluating and Selecting Big Data Technologies

Companies are spending millions on new big data technologies, and the market for such tools is expanding rapidly. In recent years, however, the IT industry has caught on to big data and analytics potential. The trending technologies include the following:

* Hadoop Ecosystem
* Apache Spark
* NoSQL Databases
* R Software
* Predictive Analytics
* Prescriptive Analytics

**What is Big Data?**

Big Data refers to massive amounts of data produced by different sources like social media platforms, web logs, sensors, IoT devices, and many more. It can be either structured (like tables in DBMS), semi-structured (like XML files), or unstructured (like audios, videos, images).

Traditional database management systems are not able to handle this vast amount of data.

Big Data helps companies to generate valuable insights.

Companies use Big Data to refine their marketing campaigns and techniques. Companies use it in machine learning projects to train machines, predictive modeling, and other advanced analytics applications.

We can’t equate big data to any specific data volume. Big data deployments can involve terabytes, petabytes, and even exabytes of data captured over time.

**Why to use Big Data?**

Big Data initiatives were rated as “extremely important” to 93% of companies. Leveraging a Big Data analytics solution helps organizations to unlock the strategic values and take full advantage of their assets.

It helps organizations:

* To understand Where, When and Why their customers buy
* Protect the company’s client base with improved loyalty programs
* Seizing cross-selling and upselling opportunities
* Provide targeted promotional information
* Optimize Workforce planning and operations
* Improve inefficiencies in the company’s supply chain
* Predict market trends
* Predict future needs
* Make companies more innovative and competitive
* It helps companies to discover new sources of revenue

Companies are using Big Data to know what their customers want, who are their best customers, why people choose different products. The more a company knows about its customers, the more competitive it becomes.

We can use it with Machine Learning for creating market strategies based on predictions about customers. Leveraging big data makes companies customer-centric.

Companies can use Historical and real-time data to assess evolving consumers’ preferences. This consequently enables businesses to improve and update their marketing strategies which make companies more responsive to customer needs.

**Differences Between Business Intelligence and Big Data?**

Below is the list of items, explain the differences between the Business Intelligence and Big Data

Both BI and Big data goal is to help the business to make good decisions by analyzing the huge datasets to expand the business and optimizing the cost.

* This data analysis not only enables decision making but also involves an active part in the development of strategies and methods that make sure the success of organizations. This data analysis can be called “Business Intelligence”, whereas “Big Data” is a relatively new term for Business intelligence.
* Since the times of BI, the volumes of data sets become incredibly large, the best example we can consider is social media. As the result, more effort and strategies should be applied to tackle with them and make them useful for successful business.
* Business Intelligence helps in finding the answers to the business questions we know, whereas Big Data helps us in finding the questions and answers that we didn’t know before.
* Although Business Intelligence and Big Data are two technologies used to analyze data sets to helps organizations in the decision-making process, there is differences present between them. They both differ in the way they analyze the data.
* Business Intelligence is based on the principle of combining all business data sets into a central server, this data will be analyzed in offline mode, after saving the information in a platform or environment called Data Warehouse. The data sets are structured in a relational database with additional indexes and forms of access to the tables in the warehouse.
* Whereas in the Big Data environment, data is stored on a distributed file system (e.g. HDFS), rather than storing on a central server. Data will be distributed across the worker nodes for easy processing. Distributed File System is much safer and flexible.
* BI solutions carry the data to the processing functions, whereas Big Data solutions take the processing functions to the data sets. Since the analysis is positioned around the information (Data), it is simpler to handler lager amounts.
* BI solutions are more towards the structured data, whereas Big Data tools can process and analyze data in different formats, both structured and unstructured.
* Big Data solutions can process the historical data and also data coming from real-time sources, whereas in Business Intelligence, it processes the historical data sets.
* Big Data technology uses parallel processing concepts (Map reducing algorithm), which improves the speed of analyzing and processing the data sets by distributing jobs into several parallel execution processes, at the end the results are combined and shown, this makes analyzing the large volumes easier.

**Business Intelligence vs Big Data**

|  |  |  |
| --- | --- | --- |
| **Comparison of Objectives** | **Business Intelligence** | **Big Data** |
| **Purpose** | The purpose of Business Intelligence is to help the business to make better decisions. Business Intelligence helps in delivering accurate reports by extracting information directly from the data source. | The main purpose of Big Data is to capture, process, and analyze the data, both structured and unstructured to improve customer outcomes. |
| **EcoSystem / Components** | Operation systems, ERP databases, Data Warehouse, Dashboard etc. | Hadoop, Spark, R Server, hive, HDFS etc. |
| **Tools** | Below is the list of tools used for business intelligence. These tools enable a business to collate, analyze and visualize data, which can be used in making better business decisions and to come up with good strategic plans.   * Tableau * Qlik Sense * Online analytical processing  (OLAP) * Sisense * Data Warehousing * Digital Dashboards and Data mining * Microsoft Power BI * Google Analytics etc | Below is the list of tools used in Big Data. These tools or frameworks store a large amount of data and process them to get insights from data to make good decisions for the business.   * Hadoop * Spark * Hive * Polybase * Presto * Cassandra * Plotly * Cloudera * Storm etc |
| **Characteristics/ Properties** | Big data can be described by some characteristics such as Volume, Variety, Variability, Velocity, and Veracity. | Below are the six features of Business Intelligence Location intelligence, Executive Dashboards, “what if” analysis,  Interactive reports, Metadata layer, and Ranking reports |
| **Benefits** | Below is the list of benefits of Business Intelligence   * Helps in making better business decisions * Faster and more accurate reporting and analysis * Improved data quality * Reduced costs * Increase revenues * Improved operational efficiency etc. | Below is the list of benefits of Big Data   * Better Decision making * Fraud detection * Storage, mining, and analysis of data * Market prediction &and forecasting * Improves the service * Helps in implementing the new strategies * Keep up with customer trends * Cost savings * Better sales insights, which helps in increasing revenues  etc |
| **Applied Fields** | Social media, Healthcare, Gaming Industry, Food Industry etc | The banking sector,  Entertainment, and Social media, Healthcare, Retail and wholesale etc |

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Unit – 2:**

**Big data Analytics**

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**What is and isn’t big data analytics?**

**Big Data analytics** is the process of finding patterns, trends, and relationships in massive datasets that can’t be discovered with traditional data management techniques and tools.

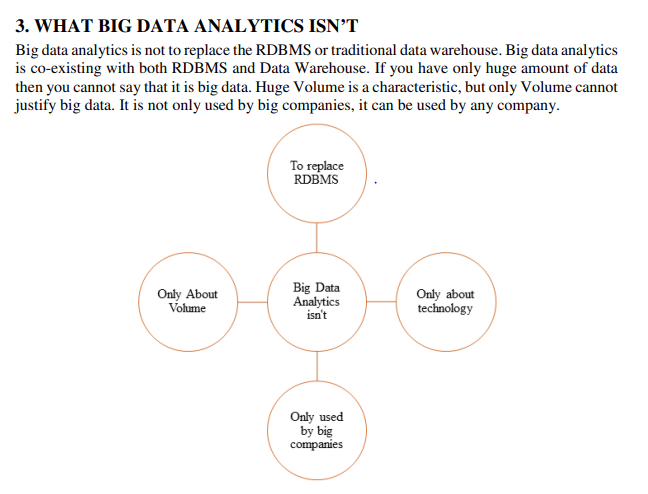
The best way to understand the idea behind Big Data analytics is to put it against regular data analytics.

* Traditional approach. The analytics commonly takes place after a certain period of time or event. If you are an owner of an online shop, you may look at the data accumulated during a week and then analyze it. For instance, you calculate which customers used discount certificates that were sent to them by email.
* Big Data. The analytics usually happens in real-time ‒ as data is being generated ‒ and discoveries are presented almost instantaneously. Say, you operate a fleet of 100 trucks and you need to know the exact location of each as well as route delays in real-time.

Data generated from various sources including sensors, log files and social media, you name it, can be utilized both independently and as a supplement to existing transactional data many organizations already have at hand. Besides, it is not just business users and analysts who can use this data for advanced analytics but also [data science teams](https://www.altexsoft.com/blog/datascience/how-to-structure-data-science-team-key-models-and-roles/) that can apply Big Data to build predictive ML projects.

There are four key types of Big Data analytics singled out.

* **Descriptive analytics** is a common kind of analytics that allows you to find out what happened and when.
* **Diagnostic analytics** explains why and how something happened by identifying patterns and relationships in available data.
* **Predictive analytics**uses historical data to uncover patterns and make predictions on what’s likely to happen in the future.
* **Prescriptive analytics** provides specific recommendations on what should be done better.



**1) Big Data isn’t a simple and efficient fix for complex problems**

**2) Big Data isn’t a solution you can lead with**

**3) Big Data isn’t “BI on steroids”**

**4) Big Data isn’t “a solution”**

**5) Big Data doesn’t lend itself well to “low hanging fruit”**

## Why hype around big data analytics?

Big data analytics helps organizations harness their data and use it to identify new opportunities. That, in turn, leads to smarter business moves, more efficient operations, higher profits and happier customers. Businesses that use big data with advanced analytics gain value in many ways, such as:

1. **Reducing cost.** Big data technologies like cloud-based analytics can significantly reduce costs when it comes to storing large amounts of data (for example, a data lake). Plus, big data analytics helps organizations find more efficient ways of doing business.
2. **Making faster, better decisions.** The speed of in-memory analytics – combined with the ability to analyze new sources of data, such as streaming data from IoT – helps businesses analyze information immediately and make fast, informed decisions.
3. **Developing and marketing new products and services.** Being able to gauge customer needs and customer satisfaction through analytics empowers businesses to give customers what they want, when they want it. With big data analytics, more companies have an opportunity to develop innovative new products to meet customers’ changing needs.

**Classification of Analytics?**

Analytics is the process of breaking the problem into simpler parts based on data to take decision. Analytics is not a tool. It is only the way of thinking.

## Different Types/Classifications of Data Analytics

**Descriptive Analytics**

**Predictive Analytics**

**Prescriptive Analytics**

**Descriptive Analytics - What Happened?**

**Descriptive Analytics** uses the existing information from the past to understand decision in present and helps to decide an effective source os action in future.

**Example**:- Large Business Organization

**Predictive Analytics - What Could Happen?**

As you might’ve guessed from the title - predictive analytics is designed to foresee:

* what the future holds (to a certain degree)
* show a variety of possible outcomes

it works by identifying patterns and compares with Historical data and then using statistics to make inferences about the future.

**Prescriptive Analytics - What Should We Do?**

It does not only say “what is happening present” but also estimates “what might happen in future” and most importantly “what to do” about it.

Example:- Research Centers

**Importance of Big Data Analytics**

**Reactive-Business Intelligence**: What does Busine Intelligence (BI) help us with? It allows the businesses to make faster and better decisions by providing the right information to the right person at the right time in the right format. Its about analysis of the past or historical data and then displaying the findings of the analysis or reports in the form of enterprise dashboards, alerts, notifications, etc. It has support for both pre-specified reports as well as ad hoc querying.

**Reactive-Big Data Analytics**: Here the analysis is done on huge datasets but the approach is still reactive as it is still based on static data.

**Proactive-Analytics**: This is to support futuristic decision making by use of data mining predictive modelling, text mining and statistical analysis on. This analysis is not on big data as it still the traditional database management practices on big data and therefore has severe limitations on the storage capacity and the processing capability.

**Proactive-Big Data Analytics**: This is filtering through terabytes, petabytes, exabytes of information to filter out the relevant data to analyze. This also includes high performance analytics to gain rapid insights from big data and the ability to solve complex problems using more data.

**Top Challenges facing Big Data**

**1.Scale**: Storage (RDBMS (Relational Database Management System) or NoSQL (Not only SQL)) is one major concern that needs to be addressed to handle the need for scaling rapidly and elastically. The need of the hour is a storage that can best withstand the attack of large volume, velocity and variety of big data. Should you scale vertically or should you scale horizontally?

2.**Security**: Most of the NoSQL big data platforms have poor security mechanisms (lack of proper authentication and authorization mechanisms) when it comes to safeguarding big data. A spot that cannot be ignored given that big data carries credit card information, personal information and other sensitive data.

**3. Schema**: Rigid schemas have no place. We want the technology to be able to fit our big data and not the other way around. The need of the hour is dynamic schema. Static (pre- defined schemas) are obsolete.

**4. Continuous availability**: The big question here is how to provide 24/7 support because almost all RDBMS and NoSQL big data platforms have a certain amount of downtime built in.

**5. Consistency**: Should one opt for consistency or eventual consistency? Partition tolerant: How to build partition tolerant systems that can take care of both hardware and software failures?

**6. Data quality**: How to maintain data quality-data accuracy, completeness, timeliness, etc.? Do we have appropriate metadata in place?

**Technologies needed to meet challenges of big data.**

**(Big Data Analytics Softwares**

**1.R-Language**: R is a Programming Language and free software environment for Statistical Computing and Graphics. The R language is widely used among Statisticians and Data Miners for developing Statistical Software and majorly in Data Analysis.

Developed by: R-Foundation in the year 2000 29th

. Written in: Fortran

Current stable version: R-3.6.0

Used Companies: Bank of America, AmericanExpress, Barclays, etc

**2. Spark**: Spark provides In-Memory Computing capabilities to deliver Speed, a Generalized Execution Model to support a wide variety of applications, and Java, Scala, and Python APIs for ease of development

Developed by: Apache Software Foundation

. Written in: Java, Scala, Python, R .

Current stable version: Apache Spark 2.4.3

Used Companies: Amazon, Oracle, Cisco, etc

**3. Hadoop**: Hadoop Framework was designed to store and proce data in a Distributed Data Processing Environment wi commodity hardware with a simple programming model. It can Ston and Analyse the data present in different machines with High Speed and Low Costs.

• Developed by: Apache Software Foundation in the yea 2011 10th of Dec.

Written in: JAVA

Current stable version: Hadoop 3.11

Used Companies: Microsoft, Intel, IBM, etc

**4. MongoDB**: The NoSQL Document Databases like MongoDB, offer a direct alternative to the rigid schema used Relational Databases. This allows MongoDB to offer Flexibility whit handling a wide variety of Datatypes at large volumes and across Distributed Architectures.

Developed by: MongoDB in the year 2009 11th of Feb

Written in: C++, Go, JavaScript, Python ⚫

Current stable version: MongoDB 4.0.10

Used Companies: MongoDB, MySql, Microsoft SQLserver, etc

**5.Elastisearch**: Elasticsearch is a Search Engine based on the Lucene Library. It provides a Distributed, MultiTenant-capable, Full- Text Search Engine with an HTTP Web Interface and Schema-free JSON documents.

•Developed by: Elastic NV in the year 2012.

. Written in: JAVA

Current stable version: ElasticSearch 7.1

Used Companies: Accenture, NetFlix, in, stackoverflow. etc

**6.BlockChain**: BlockChain is used in essential functions such as payment, escrow, and title can also reduce fraud, increase financial privacy, speed up transactions, and internationalize markets.

Developed by: Bitcoin

. Written in: JavaScript, C++, Python .

Current stable version: Blockchain 4.0

Used Companies: Alibaba.con, facebook, Oracle, etc

**7. Splunk**: Splunk captures, Indexes, and correlates Real-time data ina Searchable Repository from which it can generate Graphs, Reports, Alerts, Dashboards, and Data Visualizations. It is also used for Application Management, Security and Compliance, as well as Business and Web Analytics.

Developed by: Splunk INC in the year 2014 6th May

Written in: AJAX, C++, Python, XML

Current stable version: Splunk 7.3

Used Companies: QRadar, Trustwave, ILabs, etc.

**8. Kafka**: Apache Kafka is a Distributed Streaming platform. A streaming platform has Three Key Capabilities that are as follows:

Publisher

Subscriber

Consumer

This is similar to a Message Queue or an Enterprise Messaging System. • Developed by: Apache Software Foundation in the year 2011

Written in: Scala, JAVA

⚫ Current stable version: Apache Kafka 2.2.0

Used Companies: NetFlix, Yahoo, Twitter, Spotify, etc.

**Unit – 3:**

**Introduction to R and getting started with R**

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-III**

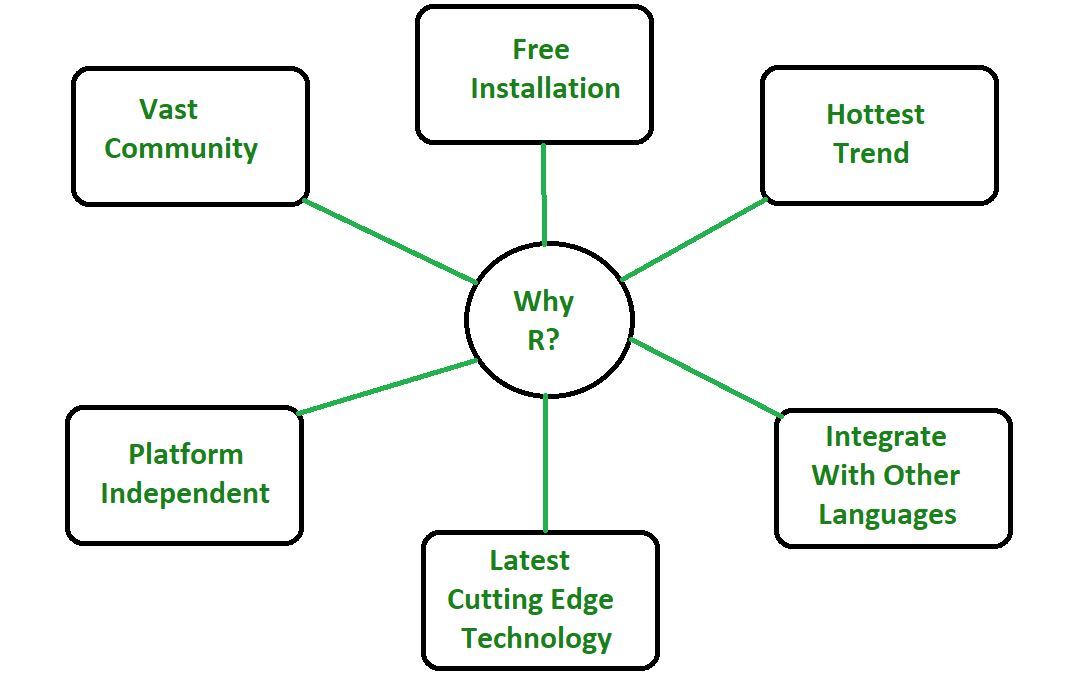
**Introduction to R and getting started with R**

**What is R? Why R?**

R is an open-source programming language that is widely used as a statistical software and data analysis tool. R generally comes with the Command-line interface. R is available across widely used platforms like Windows, Linux, and macOS. Also, the R programming language is the latest cutting-edge tool.

It was designed by **Ross Ihaka and Robert Gentleman** at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R programming language is an implementation of the S programming language. It also combines with lexical scoping semantics inspired by Scheme. Moreover, the project conceives in 1992, with an initial version released in 1995 and a stable beta version in 2000.

#### Why R Programming Language?



* R programming is used as a leading tool for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
* It’s a platform-independent language. This means it can be applied to all operating system.
* It’s an open-source free language. That means anyone can install it in any organization without purchasing a license.
* R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
* The R programming language has a vast community of users and it’s growing day by day.
* R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.

**Advantages of R:**

* R is the most comprehensive statistical analysis package. As new technology and concepts often appear first in R.
* As R programming language is an open source. Thus, you can run R anywhere and at any time.
* R programming language is suitable for GNU/Linux and Windows operating system.
* R programming is cross-platform which runs on any operating system.
* In R, everyone is welcome to provide new packages, bug fixes, and code enhancements.

**Disadvantages of R:**

* In the R programming language, the standard of some packages is less than perfect.
* Although, R commands give little pressure to memory management. So R programming language may consume all available memory.
* In R basically, nobody to complain if something doesn’t work.
* R programming language is much slower than other programming languages such as Python and MATLAB.

**Explain the advantages of R over other programming languages?**

Advanced programming languages like Python also suppon statistical computing and data visualization along with traditional computer programming. However, R wins the race over Python and similar languages because of the following two advantages.

1. Python needs third party extensions and support for data visualization and statistical computing. However, R does not require any such support extensively. For example, the Im function is present for linear regression analysis and data analysis in both Python and R. In R, data can be easily passed through the function and the function will return an object with detailed information about the regression. The function can also return information about the standard errors, eoefficients, residual values and so on. When Im function is called in the Python environment, it will duplicate the functionalities using third party libraries such as SciPy, NumPy and so on. Hence, R can do the same thing with a single line of code instead of taking support from third party libraries.

# 2. R has the fundamental data type, i.e., a vector that can be organized and aggregated in different ways even though the core is the same. Vector data type imposes some limitations on the language as this is a rigid type. However, it gives a strong logical base to R. based on the vector type, R uses the concept of data frames that are like matrix with attributes and internal data structure similar to spreadsheets or relational database. Hence, R follows a column-wise data structure based on the aggregation of vectors.

# R Data Types?

Like other programming languages, R also makes use of variables store varied information. This means that when variables are created, locations are reserved in the computer's memory to hold the related values. The number of locations or size of memory reserved is determined by the data type of the variable Following are data types supported by R:

1.Logical

2. Numeric

(a) Integer

3. Character

4. Double

5. Complex

6. Raw

**1. Logical**: A logical value is often created via comparison between variables. The two logical values are TRUE and FALSE, TRUE can be coerced to 1 and FALSE can be coerced to 0.

Example:

x = 1;

y = 2

z = x > Y

z # prints the logical value

class (z) # displays the data type of z

as.logical (1) # coerced to TRUE

as.logical (0) # coerced to FALSE

Output:

[1] FALSE

[1] "logical"

[1] TRUE

[1] FALSE

class () function is can be used to know the data type of variable.

**2. Numeric**: Decimal values are called numeric in R. If we assign a decimal value to a variable x as follows, x will be of numeric type. Furthermore, even if we assign an integer to a variable y, it is still being saved as numeric value and not an integer.

Example:

X=10.9

X

Y=5

Y

class (x) # displays the data type of X

class (y) # displays the data type of Y

as.numeric(x) #checks whether x is numer data type or not

as.numeric(y) #checks whether y is numeric data type or not

**Output**:

[1] 10.9

[1] 5

[1] "numeric"

[1] "numeric"

[1] TRUE

[1] TRUE

**a. Integer**: Integer data type is a sub class of numeric data type. We use of "L" as a suffix a numeric value in order for it to be considered an "Integer"

Example:

X=2L

Class(x) # displays the data type of X

is.numeric(2) #checks whether 2 is numeric data type or not

is.numeric(2L) #checks whether 2L is numeric data type or not

is. Integer (2) #checks whether 2 is integer datatype or not

is. integer (2L) #checks whether 2L is integer datatype or not

**Output:**

[1] "integer"

[1] TRUE

[1] TRUE

[1] FALSE

[1] TRUE

Functions such as is.numeric(), is.integer() can be used to test the data type.

**3 Charter** A character object is used to represent string values in R.We convert objects into character values with the as.character function. Both single quotes and double quotes can used to represent strings, **Example**

X="Data Analytics"

Y=as.character (7.8)

X

y

class (y)

is.character (x)

**output**:

[1] "Data Analytics"

[1] "7.8"

[1] "character"

[1] TRUE

is.character () function can be used to ascertain if a value is a character.

**4. Double** (for double precision floating point numbers): By default, numbers are of "double" type unless explicitly mentioned with an "L'suffixed to the number for it to be considered an integer.

**Example**:

typeof (76.25)

**Output**

[1] "double"

**5. Complex**: A complex value in R defined via the pure imaginary value i. A complex number will be of the form 'a+bi', where a is the real part and b is the imaginary part.

Example:

X=5+4i

class (x) #displays the data type of x

is.complex (x) #checks whether x is complex data type

y=as.complex (3) #coerced to complex

Y

**output**:

[1] 5+4i

[1] “complex”

[1] TRUE

[1] 3=0i

**6. Raw**: A raw data type specifies values as raw bytes. You can use the following methods to convert character data types to a raw datatype and vice-versa.

charToRaw ()-converts character data to raw data.

rawToChar()-converts raw data to character data

**Example:**

x-charToRaw ("Hi")

class (charToRaw("hi")

**Output:**

[1] 48 69

[1] "raw"

**Coercion in R?**

Coercion helps to convert one data type to another. For example logical"TRUE" value when converted to numeric yields "1". Likewise logical "FALSE" value yields "0".

> as.numeric(TRUE)

[1] 1

> as.numeric (FALSE)

[1] 0

Numeric 5 can be converted to character 5 using as.character()

> as.character (5)

[1] "5"

> as. integer (5.5)

[1] 5

On converting character, "hi" to numeric data type, the as.numeric() returns NA (unknown)

> as.numeric ("hi")

[1] NA

Warning message:

NA introduced by coercion

**Explain about ls () function with suitable example?**

The ls() function is used to list all the objects in the working environment.

**Syntax:**

ls ()

**Example**:

Height <- 4

Width <- 5

Area <- Height\* Width

ls()

**Output**:

[1] "Area" "Height" "Width"

Ls() is also useful to clean the environment before running a code. Execute the rm() function as shown to clean up the environment.

**Example:**

rm(list-ls())

1s()

character (0)

**Write about comments in R?**

Comments are included in our R program for better understanding. They are ignored by the interpreter while executing the actual program. Single comment is written using # at the beginning of the statement as follows:

#This is my First Program in R

R does not support multi-line comment as in C or Python.

**Expression in R?**

**expression()** function in [R Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is used to create an expression from the values passed as argument. It creates an object of the expression class.

***Syntax:****expression(character)*

***Parameters:******character:****Expression, like calls, symbols, constants*

**Example 1:**

|  |
| --- |
| # R program to create an expression  # Calling expression() Function  x <- expression(2 \* 3)  x    # Printing value of the expression  eval(x) |

**Output:**

expression(2\*3)

[1] 6

## ****Variables in R Programming****

R does not have a command for declaring a variable. A variable is created the moment you first assign a value to it. To assign a value to a variable, use the **<-** sign. To output (or print) the variable value, just type the variable name:

### Example

name <- "John"  
age <- 40  
name   # output "John"  
age    # output 40

From the example above, name and age are **variables**, while "John" and 40 are **values**.

A variable is a name given to a memory location, which is used to store values in a computer program. Variables in R programming can be used to store numbers (real and complex), words, matrices, and even tables. R is a dynamically programmed language which means that unlike other programming languages, we do not have to declare the data type of a variable before we can use it in our program.  
**For a variable to be valid, it should follow these rules**

* It should contain letters, numbers, and only dot or underscore characters.
* It should not start with a number (eg:- 2iota)
* It should not start with a dot followed by a number (eg:- .2iota)
* It should not start with an underscore (eg:- \_iota)
* It should not be a reserved keyword.

**Control Structures in R?**

## Conditions and If Statements

R supports the usual logical conditions from mathematics:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

These conditions can be used in several ways, most commonly in "if statements" and loops.

## The if Statement

An "if statement" is written with the if keyword, and it is used to specify a block of code to be executed if a condition is TRUE:

### Example

a <- 33  
b <- 200  
if (b > a)

{  
  print("b is greater than a")  
}

In this example we use two variables, a and b, which are used as a part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

R uses curly brackets { } to define the scope in the code.

## Else If

The else if keyword is R's way of saying "if the previous conditions were not true, then try this condition":

### Example

a <- 33  
b <- 33  
if (b > a)

{  
  print("b is greater than a")  
}

else

if (a == b)

{  
  print ("a and b are equal")  
}

In this example a is equal to b, so the first condition is not true, but the else if condition is true, so we print to screen that "a and b are equal".

You can use as many else if statements as you want in R.

# R Nested If

## Nested If Statements

You can also have if statements inside if statements, this is called nested if statements.

### Example

x <- 41  
if (x > 10)

{  
  print("Above ten")  
  if (x > 20)

{  
    print("and also above 20!")  
  } else {  
    print("but not above 20.")  
  }  
}

else

{  
  print("below 10.")  
}

## R Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

R has two loop commands:

* while loops
* for loops

## R While Loops

With the while loop we can execute a set of statements as long as a condition is TRUE:

### Example

Print i as long as i is less than 6:

i <- 1  
while (i < 6)

{  
  print(i)  
  i <- i + 1  
}

In the example above, the loop will continue to produce numbers ranging from 1 to 5. The loop will stop at 6 because 6 < 6 is FALSE.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

**Note:** remember to increment i, or else the loop will continue forever.

## Break

With the break statement, we can stop the loop even if the while condition is TRUE:

### Example

Exit the loop if i is equal to 4.

i <- 1  
while (i < 6)

{  
  print(i)  
  i <- i + 1  
  if (i == 4)

{  
    break  
  }  
}

The loop will stop at 3 because we have chosen to finish the loop by using the break statement when i is equal to 4 (i == 4).

## Next

With the next statement, we can skip an iteration without terminating the loop:

### Example

Skip the value of 3:

i <- 0  
while (i < 6) {  
  i <- i + 1  
  if (i == 3) {  
    next  
  }  
  print(i)  
}

When the loop passes the value 3, it will skip it and continue to loop.

# R For Loop

## For Loops

A for loop is used for iterating over a sequence:

### Example

for (x in 1:10) {  
  print(x)  
}

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

## Break

With the break statement, we can stop the loop before it has looped through all the items:

### Example

Stop the loop at "cherry":

fruits <- list("apple", "banana", "cherry")  
for (x in fruits) {  
  if (x == "cherry") {  
    break  
  }  
  print(x)  
}

The loop will stop at "cherry" because we have chosen to finish the loop by using the break statement when x is equal to "cherry" (x == "cherry").

# R Nested Loops

## Nested Loops

It is also possible to place a loop inside another loop. This is called a **nested loop**:

### Example

Print the adjective of each fruit in a list:

adj <- list("red", "big", "tasty")  
fruits <- list("apple", "banana", "cherry")  
  for (x in adj) {  
    for (y in fruits) {  
      print(paste(x, y))  
  }  
}

# R Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

## Creating a Function

To create a function, use the function() keyword:

### Example

my\_function <- function()

{  # create a function with the name my\_function  
  print("Hello World!")  
}

## Call a Function

To call a function, use the function name followed by parenthesis, like **my\_function()**:

### Example

my\_function <- function()

{  
  print("Hello World!")  
}  
**my\_function()** # call the function named my\_function

## Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

## Return Values

To let a function return a result, use the return() function:

### Example

my\_function <- function(x) {  
  return (5 \* x)  
}  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

The output of the code above will be:

[1] 15  
[1] 25  
[1] 45

# R Arrays

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the **array()** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

## Example

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2))

print(result)

When we execute the above code, it produces the following result −

, , 1

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

[1,] 5 10 13

[2,] 9 11 14

[3,] 3 12 15

, , 2

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

[1,] 5 10 13

[2,] 9 11 14

[3,] 3 12 15

## Accessing Array Elements

# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

column.names <- c("COL1","COL2","COL3")

row.names <- c("ROW1","ROW2","ROW3")

matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,

column.names, matrix.names))

# Print the third row of the second matrix of the array.

print(result[3,,2])

# Print the element in the 1st row and 3rd column of the 1st matrix.

print(result[1,3,1])

# Print the 2nd Matrix.

print(result[,,2])

When we execute the above code, it produces the following result −

COL1 COL2 COL3

3 12 15

[1] 13

COL1 COL2 COL3

ROW1 5 10 13

ROW2 9 11 14

ROW3 3 12 15

# R Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

**Syntax**

The basic syntax for creating a matrix in R is −

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

Following is the description of the parameters used −

* **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

# Create a matrix  
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)  
# Print the matrix  
thismatrix

**Note:** Remember the c() function is used to concatenate items together.

You can also create a matrix with strings:

### Example

thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)  
thismatrix

## Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

# Define the column and row names.

rownames = c("row1", "row2", "row3", "row4")

colnames = c("col1", "col2", "col3")

# Create the matrix.

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

# Access the element at 3rd column and 1st row.

print(P[1,3])

# Access the element at 2nd column and 4th row.

print(P[4,2])

# Access only the 2nd row.

print(P[2,])

# Access only the 3rd column.

print(P[,3])

When we execute the above code, it produces the following result −

[1] 5

[1] 13

col1 col2 col3

6 7 8

row1 row2 row3 row4

5 8 11 14

# R Vectors

## Vectors

A vector is simply a list of items that are of the same type.

To combine the list of items to a vector, use the c() function and separate the items by a comma.

In the example below, we create a vector variable called **fruits**, that combine strings:

### Example

# Vector of strings  
fruits <- c("banana", "apple", "orange")  
# Print fruits  
fruits

In this example, we create a vector that combines numerical values:

### Example

# Vector of numerical values  
numbers <- c(1, 2, 3)  
# Print numbers  
numbers

To create a vector with numerical values in a sequence, use the : operator:

### Example

# Vector with numerical values in a sequence  
numbers <- 1:10  
numbers

You can also create numerical values with decimals in a sequence, but note that if the last element does not belong to the sequence, it is not used:

### Example

# Vector with numerical decimals in a sequence  
numbers1 <- 1.5:6.5  
numbers1  
# Vector with numerical decimals in a sequence where the last element is not used  
numbers2 <- 1.5:6.3  
numbers2

Result:

[1] 1.5 2.5 3.5 4.5 5.5 6.5  
[1] 1.5 2.5 3.5 4.5 5.5

In the example below, we create a vector of logical values:

### Example

# Vector of logical values  
log\_values <- c(TRUE, FALSE, TRUE, FALSE)  
log\_values  
Vector Length

To find out how many items a vector has, use the length() function:

### Example

fruits <- c("banana", "apple", "orange")  
length(fruits)

## Sort a Vector

To sort items in a vector alphabetically or numerically, use the sort() function:

### Example

fruits <- c("banana", "apple", "orange", "mango", "lemon")  
numbers <- c(13, 3, 5, 7, 20, 2)  
sort(fruits)  # Sort a string  
sort(numbers) # Sort numbers

## Access Vectors

You can access the vector items by referring to its index number inside brackets []. The first item has index 1, the second item has index 2, and so on:

### Example

fruits <- c("banana", "apple", "orange")  
# Access the first item (banana)  
fruits[1]

You can also access multiple elements by referring to different index positions with the c() function:

### Example

fruits <- c("banana", "apple", "orange", "mango", "lemon")  
# Access the first and third item (banana and orange)  
fruits[c(1, 3)]

You can also use negative index numbers to access all items except the ones specified:

### Example

fruits <- c("banana", "apple", "orange", "mango", "lemon")  
# Access all items except for the first item  
fruits[c(-1)]

## Change an Item

To change the value of a specific item, refer to the index number:

### Example

fruits <- c("banana", "apple", "orange", "mango", "lemon")  
# Change "banana" to "pear"  
fruits[1] <- "pear"  
# Print fruits  
fruits

## Repeat Vectors

To repeat vectors, use the rep() function:

### Example

Repeat each value:

repeat\_each <- rep(c(1,2,3), each = 3)  
repeat\_each

### Example

Repeat the sequence of the vector:

repeat\_times <- rep(c(1,2,3), times = 3)  
repeat\_times

### Example

Repeat each value independently:

repeat\_indepent <- rep(c(1,2,3), times = c(5,2,1))  
repeat\_indepent

## Generating Sequenced Vectors

One of the examples on top, showed you how to create a vector with numerical values in a sequence with the : operator:

### Example

numbers <- 1:10  
numbers

To make bigger or smaller steps in a sequence, use the seq() function:

### Example

numbers <- seq(from = 0, to = 100, by = 20)  
numbers

**Note:** The seq() function has three parameters: from is where the sequence starts, to is where the sequence stops, and by is the interval of the sequence.

# R Packages?

Various R functions, sample data, and compile codes are grouped to create R Packages, that are stored under a directory called “library”, in the R environment. Some sets of packages are installed with R installation, while other packages can be added as and when required. The packages installed with R installation are the default packages that are obtainable whenever the R console starts. The packages installed later on require explicit loading. They can then be utilised as required in programming with R or R codes.

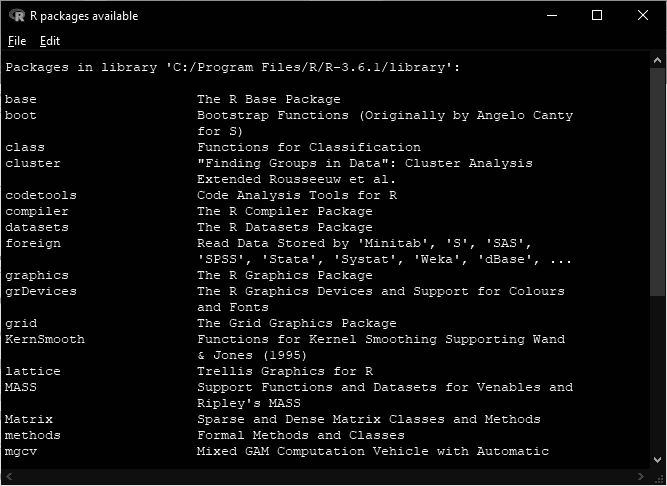
### Functions to check Available R Packages:

* **libPaths():**It is important to find the library locations to look for the available R Packages. This is where libPaths() function is utilised.

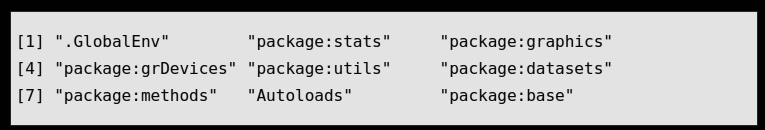
### Functions to get the list of all the installed packages:

* **library():**The list of all the installed packages can be made available using the library() function in R.

**Packages in library ‘C:/Program Files/R/R-3.6.1/library’:**



### search()****:****The packages presently installed in the R environment can also be viewed using the search() function in R.



### Installing a New Package in R:

New packages can be installed in R using either of the below two techniques:

* Installing packages directly from the CRAN directory.
* Installing packages manually after downloading the package to the local system.

### Installing R packages directly from the CRAN directory:

The packages can be installed directly from CRAN webpage to the R environment, using the below command.

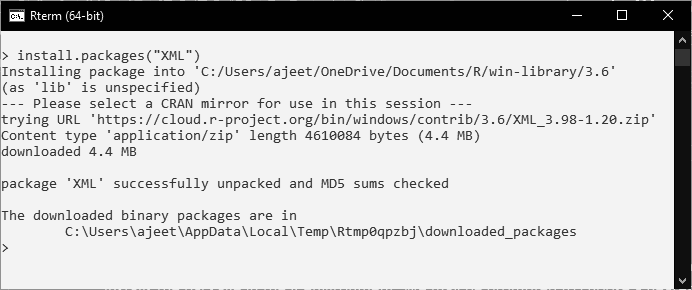
install.packages(“Package Name”)

**Example:**For installing XML package:

install.packages(“XML”)

**Note::**If prompted to select the nearest mirror, select the mirror which is appropriate to your location.

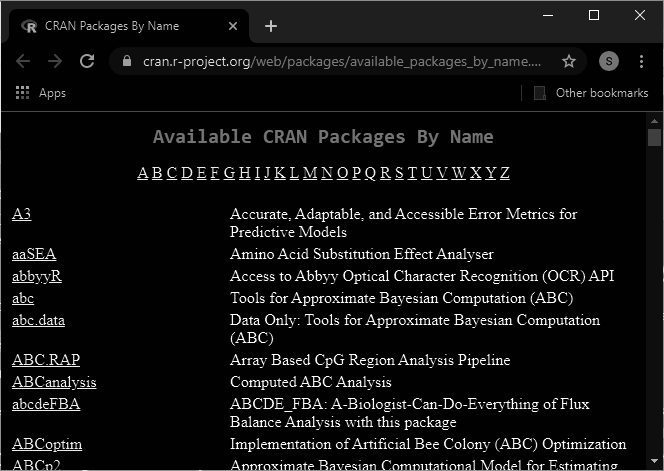
**Output:**



### Installing packages manually after downloading the package to the local system:

To install the R packages manually, follow the below steps:

* Go to the link <https://cran.r-project.org/web/packages/available_packages_by_name.html>.
* Download the desired package.
* The package will be downloaded as a .zip file.
* Locate the file in your local system.



Use the below command, after downloading the package:

install.packages(file\_name\_with\_path, repos = NULL, type = “source”)

**Example:**For installing the package named “XML”:

install.packages(“C:\Users\example\OneDrive\Desktop\graphics\xml2\_1.2.2.zip”, repos = NULL, type = “source”)

### Loading Packages to the Library:

Before using the installed packages in your code, you need to load them into the current R environment. For this purpose, you can make use of the below command:

library(“package Name”, lib.loc = “path to library”)

**UNIT-IV**

**Exploring Data in R**

**Data frames in R?**

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 from each column.

Following are the characteristics of a 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.

## Create Data Frame

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

)

# Print the data frame.

print(emp.data)

When we execute the above code, it produces the following result −

emp\_id emp\_name salary start\_date

1 1 Rick 623.30 2012-01-01

2 2 Dan 515.20 2013-09-23

3 3 Michelle 611.00 2014-11-15

4 4 Ryan 729.00 2014-05-11

5 5 Gary 843.25 2015-03-27

**Accessing a Data Frame:-**

To access the second column “EName”, we type the following command at thr R promt.

>Employee[2]

emp\_name

1 Rick

2 Dan

3 Michelle

4 Ryan

5 Gary

To access the first and second column, “ENo” and “EName”, we type the following command at the R prompt.

>Employee[1:2]

emp\_id emp\_name

1 1 Rick

2 2 Dan

3 3 Michelle

4 4 Ryan

5 5 Gary

# Ordering/How to Sort a DataFrame in R ?

**Methods to sort a dataframe:**

1. order() function (increasing and decreasing order)
2. arrange() function from dplyr package
3. setorder() function from data.table package

#### ****Method 1: Using order() function****

This function is used to sort the dataframe based on the particular column in the dataframe

***Syntax:****order(dataframe$column\_name,decreasing = TRUE))*

***where***

* *dataframe is the input dataframe*
* *Column name is the column in the dataframe such that dataframe is sorted based on this column*
* *Decreasing parameter specifies the type of sorting order*

*If it is TRUE dataframe is sorted in descending order. Otherwise, in increasing order*

#### ****Method 2: Using arrange() Function .****

**Arrange()**is used to sort the dataframe in increasing order, it will also sort the dataframe based on the column in the dataframe

***Syntax:****arrange(dataframe,column)*

*where*

* *dataframe is the dataframe input*
* *column is the column name , based on this column dataframe is sorted*

We need to install dplyr package as it is available in that package

***Syntax:****install.packages(“dplyr”)*

#### ****Method 3: Using setorder() from data.table package****

**setorder**is used to sort a dataframe in the set order format.

***Syntax****: setorder(dataframe, column)*

* *Where dataframe is the input dataframe*
* *The column is the column name*

## R Functions for Data Frame

## We will explore the data held in the data frames with the help of the following functions:

* 1. Dim()
     1. Nrow()
     2. Ncols()
  2. Str()
  3. Summary()
  4. Names()
  5. Head()
  6. Tails()
  7. Edit()

1.dim() Function: This function is used to obtain the dimensions of a data frame. The output of this function returns the number of rows and columns.

Example:

> dim(Employee)

[1] 5 4

nrow() Function: This function returns the number of rows in a data frame.

Example:

> nrow(Employee)

[1] 5

ncol () Function: This function returns the number of columns in data frame.

Example:

> ncol (Employee)

[1] 4

**2. str() Function**: This function compactly displays the internal structure of R objects. We will use it to display the the dataset, "Employee"

**Example:**

> str(Employee)

'data.frame": 5 obs. of 4 variables:

$ ENO : num 1000 1001 1002 1003 1004

$ Ename: Factor w/ 5 levels "Hari", "Kareem",…..

2 1 4 5 3

$ salary: num 10000 15000 20000 30000 40000

$ Address: chr "Proddatur" "Kadapa" "Anantapur" "Tirupathi"

**4.summary () Function**: This function return summaries for each column of the dataset.

This syntax uses the following basic syntax:

**summary(data)**

The following examples show how to use this function in practice.

**Example 1: Using summary() with Vector**

The following code shows how to use the **summary()** function to summarize the values in a vector:

**#define vector**

**x <- c(3, 4, 4, 5, 7, 8, 9, 12, 13, 13, 15, 19, 21)**

**#summarize values in vector**

**summary(x)**

**Min. 1st Qu. Median Mean 3rd Qu. Max.**

**3.00 5.00 9.00 10.23 13.00 21.00**

The **summary()** function automatically calculates

the following summary statistics for the vector:

* Min: The minimum value
* 1st Qu: The value of the 1st quartile (25th percentile)
* Median: The median value
* 3rd Qu: The value of the 3rd quartile (75th percentile)
* Max: The maximum value

Note that if there are any missing values (NA) in the vector, the **summary()** function will

automatically exclude them when calculating the summary statistics:

**4.Names ()Functions**: This function returns the names of the objects. This function returns the column headers for the dataset, "Employee".

**Example:**

names (Employee)

[1] "ENO" "Ename" "Salary" "Address"

**5.head () Function: This** function is used to obtain the first n observations where n is set as 6 by default

Example:

>head (Employee, n=3)

ENO Ename Salary Address

employee 1 1000 Kareem 10000 Proddatur

employee 2 1001 Hari 15000 Kadapa

Employee 3 1002 shankar 20000 Anantapur

**6. tail () Function**: This function is used to obtain the last n observations where n is set as 6 by default.

Example:

>tail (Employee, n=3)

ENO Ename Salary Address

Employee 3 1002 Shankar 20000 Anantapur

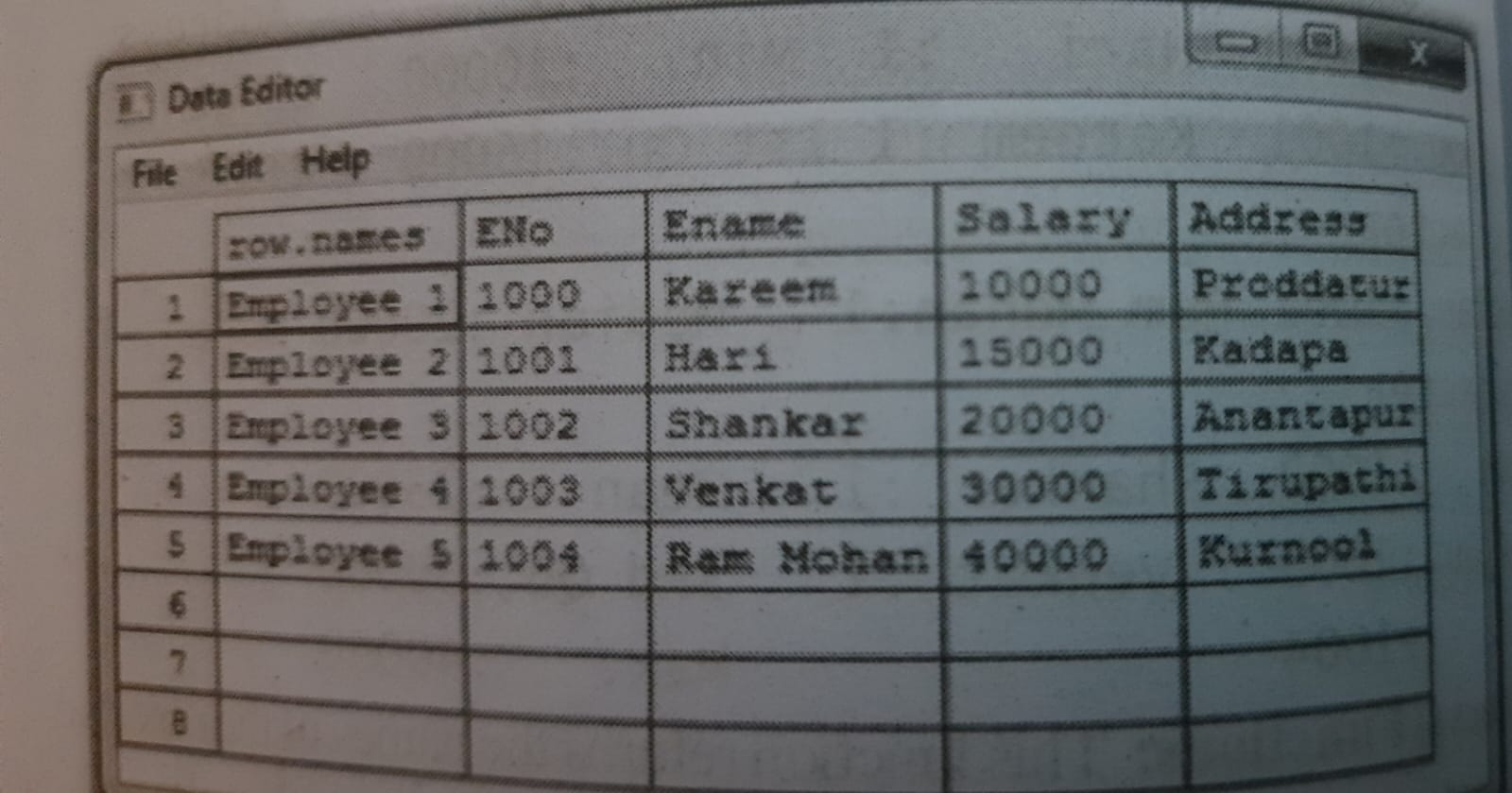
Employee 4 1003 Venkat 30000 Tirupathi

Employee 5 1004 Ram Mohan 40000 kurnool

**7.edit() Function**: This function will invoke the text editor on the R object. We will use this function to open the dataset, "Employee", in the text editor.

Example:

>edit(Employee)



To retrieve the first three rows (with all columns) form the datase "Employee", use as follow:

> Employee[1:3.]

ENO Ename Salary Address

Employee1 1000 Kareem 10000 Proddatur

Employee2 1001 Hari 15000 Kadapa

Employee3 1002 Shankar 20000 Anantapur

**Q) Explain how load data frames in R? (or) Explain how R can load data into data frames for external files?**

1. Reading from a .csv (comma separated values file)

2. Subsetting Data Frame

3.Reading from a Tab Separated Value File

4.Reading from a Table

**1. Reading from a .csv (comma separated values file)**

We have to create .csv file by the name "item.csv in the D:\ drive. It has the following data.

Itemcode, ItemCategory, ItemPrice

11001, Electronics, 700 ,

11002, Desktop supplies, 300

11003, Office Supplies, 350

Open Notepad and enter the above data. Save the file as "item.csv".

select Save as type:" as "All Files"

to load this file using the read.csv function.

>itemDataFrame<-read.csv("G:/item.csv”)

>ItemDataFrame

Itemcode ItemCategory ItemPrice

1 11001 Electronics 700

2 11002 Desktop supplies 300

3 11003 Office Supplies 350

**2 Subsetting Data Frame**: To subset the data frame and display the details of only those items

whose price is greater than or equal to 350.

>subset(ItemDataFrame, ItemPrice>=350)

Itemcode ItemCategory ItemPrice

1 11001 Electronics 700

3 11003 Office Supplies 350

To subset the data frame and display only the category to which the items belong (items whose price is greater than or equal to 350)

>subset(ItemDataFrame, ItemPrice>=350, select=c(ItemCategory)

ItemCategory

* 1. Electronics
  2. office Supplies

To subset the data frame and display only the items where the category is either "Office supplies" or "Desktop supplies"

>subset (ItemDataFrame, ItemCategory=="Office Supplies" |ItemCategory=="Desktop supplies")

Itemcode ItemCategory ItemPrice

2 11002 Desktop supplies 300

3 11003 Office Supplies 350

**3. Reading from a Tab Separated Value File**: For any file that uses a delimiter other than a comma, one can use the read.table command.

Example:

We have created a tab separated file by the name, "item-tab-sep.txt” in the D:\drive. It has the following content.

Itemcode itemQtyOnHand itemReorderLvl

11001 75 25

11002 30 25

11003 35 25

Let us load this file using the read.table function. We will read the content from the file but will not

>read.table("d:/item-tab-sep.txt",sep="\t")

V1 V2 V3

1 Itemcode itemQtyOnHand itemReorderLvl

2 11001 75 25

3 11002 30 25

4 11003 35 25

**4. Reading from a Table**: A data table can reside in a text file. The cells inside the table are separated by blank characters. An employee of a table with 4 rows and 3 columns is given as follows:

1001 Physics 85

1002 Computers 87

1003 Maths 93

1004 Stats 84

Copy and paste the table in a file named "d:/mydata.txt" with a text editor and then load the data into the workspace with the function read.

>mydata=read.table("d:/mydata.txt")

>mydata

V1 V2 V3

1 1001 Physics 85

2 1002 Computers 87

3 1003 Maths 93

4 1004 Stats 84

**UNIT-V**

**Data Visualization using R**

**Reading and Getting data into R(External Data):-**

**XML Files:-**

XML is a file format which shares both the file format and the data. It stands for Extensible Markup Language (XML). Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in xml the markup tags describe the meaning of the data contained into the file.

You can read a xml file in R using the "XML" package. This package can be installed using following command.

install.packages("XML")

## Input Data (Create a XMl file)

Create a XMl file by copying the below data into a text editor like notepad. Save the file with a **.xml** extension and choosing the file type as **all files(\*.\*)**.

<RECORDS>

<EMPLOYEE>

<ID>1</ID>

<NAME>Rick</NAME>

<SALARY>623.3</SALARY>

<STARTDATE>1/1/2012</STARTDATE>

<DEPT>IT</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>2</ID>

<NAME>Dan</NAME>

<SALARY>515.2</SALARY>

<STARTDATE>9/23/2013</STARTDATE>

<DEPT>Operations</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>3</ID>

<NAME>Michelle</NAME>

<SALARY>611</SALARY>

<STARTDATE>11/15/2014</STARTDATE>

<DEPT>IT</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>4</ID>

<NAME>Ryan</NAME>

<SALARY>729</SALARY>

<STARTDATE>5/11/2014</STARTDATE>

<DEPT>HR</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>5</ID>

<NAME>Gary</NAME>

<SALARY>843.25</SALARY>

<STARTDATE>3/27/2015</STARTDATE>

<DEPT>Finance</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>6</ID>

<NAME>Nina</NAME>

<SALARY>578</SALARY>

<STARTDATE>5/21/2013</STARTDATE>

<DEPT>IT</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>7</ID>

<NAME>Simon</NAME>

<SALARY>632.8</SALARY>

<STARTDATE>7/30/2013</STARTDATE>

<DEPT>Operations</DEPT>

</EMPLOYEE>

<EMPLOYEE>

<ID>8</ID>

<NAME>Guru</NAME>

<SALARY>722.5</SALARY>

<STARTDATE>6/17/2014</STARTDATE>

<DEPT>Finance</DEPT>

</EMPLOYEE>

</RECORDS>

## Reading XML File

The xml file is read by R using the function **xmlParse()**. It is stored as a list in R.

# Load the package required to read XML files.

library("XML")

# Also load the other required package.

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Print the result.

print(result)

When we execute the above code, it produces the following result −

1

Rick

623.3

1/1/2012

IT

2

Dan

515.2

9/23/2013

Operations

3

Michelle

611

11/15/2014

IT

4

Ryan

729

5/11/2014

HR

5

Gary

843.25

3/27/2015

Finance

6

Nina

578

5/21/2013

IT

7

Simon

632.8

7/30/2013

Operations

8

Guru

722.5

6/17/2014

Finance

### Get Number of Nodes Present in XML File

# Load the packages required to read XML files.

library("XML")

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.

rootnode <- xmlRoot(result)

# Find number of nodes in the root.

rootsize <- xmlSize(rootnode)

# Print the result.

print(rootsize)

When we execute the above code, it produces the following result −

output

[1] 8

## Details of the First Node

Let's look at the first record of the parsed file. It will give us an idea of the various elements present in the top level node.

# Load the packages required to read XML files.

library("XML")

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.

rootnode <- xmlRoot(result)

# Print the result.

print(rootnode[1])

When we execute the above code, it produces the following result −

$EMPLOYEE

1

Rick

623.3

1/1/2012

### Get Different Elements of a Node

# Load the packages required to read XML files.

library("XML")

library("methods")

# Give the input file name to the function.

result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.

rootnode <- xmlRoot(result)

# Get the first element of the first node.

print(rootnode[[1]][[1]])

# Get the fifth element of the first node.

print(rootnode[[1]][[5]])

# Get the second element of the third node.

print(rootnode[[3]][[2]])

When we execute the above code, it produces the following result −

1

IT

Michelle

**R-Web Data**

Many websites provide data for its users. For example the World Health Organization(WHO) provides reports on health and medical information in the form of CSV, txt and XML files. Using R programs, we can programmatically extract specific data from such websites. Some packages in R which are used to scrap data form the web are − "RCurl",XML", and "stringr". They are used to connect to the URL’s, identify required links for the files and download them to the local environment.

## Install R Packages

The following packages are required for processing the URL’s and links to the files. If they are not available in your R Environment, you can install them using following commands.

install.packages("RCurl")

install.packages("XML")

install.packages("stringr")

install.packages("plyr")

## Input Data

We will visit the URL [weather data](https://www.geos.ed.ac.uk/~weather/jcmb_ws/) and download the CSV files using R for the year 2015.

## Example

We will use the function **getHTMLLinks()** to gather the URLs of the files. Then we will use the function **download.file()** to save the files to the local system. As we will be applying the same code again and again for multiple files, we will create a function to be called multiple times. The filenames are passed as parameters in form of a R list object to this function.

# Read the URL.

url <- "http://www.geos.ed.ac.uk/~weather/jcmb\_ws/"

# Gather the html links present in the webpage.

links <- getHTMLLinks(url)

# Identify only the links which point to the JCMB 2015 files.

filenames <- links[str\_detect(links, "JCMB\_2015")]

# Store the file names as a list.

filenames\_list <- as.list(filenames)

# Create a function to download the files by passing the URL and filename list.

downloadcsv <- function (mainurl,filename) {

filedetails <- str\_c(mainurl,filename)

download.file(filedetails,filename)

}

# Now apply the l\_ply function and save the files into the current R working directory.

l\_ply(filenames,downloadcsv,mainurl = "http://www.geos.ed.ac.uk/~weather/jcmb\_ws/")

## Verify the File Download

After running the above code, you can locate the following files in the current R working directory.

"JCMB\_2015.csv" "JCMB\_2015\_Apr.csv" "JCMB\_2015\_Feb.csv" "JCMB\_2015\_Jan.csv"

"JCMB\_2015\_Mar.csv"

# R - Databases

R can connect easily to many relational databases like MySql, Oracle, Sql server etc. and fetch records from them as a data frame. Once the data is available in the R environment, it becomes a normal R data set and can be manipulated or analyzed using all the powerful packages and functions.

## R MySQL Package

R has a built-in package named "RMySQL" which provides native connectivity between with MySql database. You can install this package in the R environment using the following command.

install.packages("RMySQL")

## Connecting R to MySql

Once the package is installed we create a connection object in R to connect to the database. It takes the username, password, database name and host name as input.

# Create a connection Object to MySQL database.

# We will connect to the sampel database named "sakila" that comes with MySql installation.

mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = '\_\_\_\_',

host = 'localhost')

# List the tables available in this database.

dbListTables(mysqlconnection)

When we execute the above code, it produces the following result −

[1] "actor" "actor\_info"

[3] "address" "category"

[5] "city" "country"

[7] "customer" "customer\_list"

[9] "film" "film\_actor"

[11] "film\_category" "film\_list"

[13] "film\_text" "inventory"

[15] "language" "nicer\_but\_slower\_film\_list"

[17] "payment" "rental"

[19] "sales\_by\_film\_category" "sales\_by\_store"

[21] "staff" "staff\_list"

[23] "store"

## Querying the Tables

We can query the database tables in MySql using the function **dbSendQuery()**. The query gets executed in MySql and the result set is returned using the R **fetch()** function. Finally it is stored as a data frame in R.

# Query the "actor" tables to get all the rows.

result = dbSendQuery(mysqlconnection, "select \* from actor")

# Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.

data.frame = fetch(result, n = 5)

print(data.fame)

When we execute the above code, it produces the following result −

actor\_id first\_name last\_name last\_update

1 1 PENELOPE GUINESS 2006-02-15 04:34:33

2 2 NICK WAHLBERG 2006-02-15 04:34:33

3 3 ED CHASE 2006-02-15 04:34:33

4 4 JENNIFER DAVIS 2006-02-15 04:34:33

5 5 JOHNNY LOLLOBRIGIDA 2006-02-15 04:34:33

## Query with Filter Clause

We can pass any valid select query to get the result.

result = dbSendQuery(mysqlconnection, "select \* from actor where last\_name = 'TORN'")

# Fetch all the records(with n = -1) and store it as a data frame.

data.frame = fetch(result, n = -1)

print(data)

When we execute the above code, it produces the following result −

actor\_id first\_name last\_name last\_update

1 18 DAN TORN 2006-02-15 04:34:33

2 94 KENNETH TORN 2006-02-15 04:34:33

3 102 WALTER TORN 2006-02-15 04:34:33

## Updating Rows in the Tables

We can update the rows in a Mysql table by passing the update query to the dbSendQuery() function.

dbSendQuery(mysqlconnection, "update mtcars set disp = 168.5 where hp = 110")

After executing the above code we can see the table updated in the MySql Environment.

## Inserting Data into the Tables

dbSendQuery(mysqlconnection,

"insert into mtcars(row\_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)

values('New Mazda RX4 Wag', 21, 6, 168.5, 110, 3.9, 2.875, 17.02, 0, 1, 4, 4)"

)

After executing the above code we can see the row inserted into the table in the MySql Environment.

## Creating Tables in MySql

We can create tables in the MySql using the function **dbWriteTable()**. It overwrites the table if it already exists and takes a data frame as input.

# Create the connection object to the database where we want to create the table.

mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'name’,

host = 'localhost')

# Use the R data frame "mtcars" to create the table in MySql.

# All the rows of mtcars are taken inot MySql.

dbWriteTable(mysqlconnection, "mtcars", mtcars[, ], overwrite = TRUE)

After executing the above code we can see the table created in the MySql Environment.

## Dropping Tables in MySql

We can drop the tables in MySql database passing the drop table statement into the dbSendQuery() in the same way we used it for querying data from tables.

dbSendQuery(mysqlconnection, 'drop table if exists mtcars')

After executing the above code we can see the table is dropped in the MySql Environment.

# R - Excel File

Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format. R can read directly from these files using some excel specific packages. Few such packages are - XLConnect, xlsx, gdata etc. We will be using xlsx package. R can also write into excel file using this package.

## Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with required package name to install the additional packages.

install.packages("xlsx")

## Verify and Load the "xlsx" Package

Use the following command to verify and load the "xlsx" package.

# Verify the package is installed.

any(grepl("xlsx",installed.packages()))

# Load the library into R workspace.

library("xlsx")

When the script is run we get the following output.

[1] TRUE

Loading required package: rJava

Loading required package: methods

Loading required package: xlsxjars

## Input as xlsx File

Open Microsoft excel. Copy and paste the following data in the work sheet named as sheet1.

id name salary start\_date dept

1 Rick 623.3 1/1/2012 IT

2 Dan 515.2 9/23/2013 Operations

3 Michelle 611 11/15/2014 IT

4 Ryan 729 5/11/2014 HR

5 Gary 43.25 3/27/2015 Finance

6 Nina 578 5/21/2013 IT

7 Simon 632.8 7/30/2013 Operations

8 Guru 722.5 6/17/2014 Finance

Also copy and paste the following data to another worksheet and rename this worksheet to "city".

name city

Rick Seattle

Dan Tampa

Michelle Chicago

Ryan Seattle

Gary Houston

Nina Boston

Simon Mumbai

Guru Dallas

Save the Excel file as "input.xlsx". You should save it in the current working directory of the R workspace.

## Reading the Excel File

The input.xlsx is read by using the **read.xlsx()** function as shown below. The result is stored as a data frame in the R environment.

# Read the first worksheet in the file input.xlsx.

data <- read.xlsx("input.xlsx", sheetIndex = 1)

print(data)

When we execute the above code, it produces the following result −

id, name, salary, start\_date, dept

1 1 Rick 623.30 2012-01-01 IT

2 2 Dan 515.20 2013-09-23 Operations

3 3 Michelle 611.00 2014-11-15 IT

4 4 Ryan 729.00 2014-05-11 HR

5 NA Gary 843.25 2015-03-27 Finance

6 6 Nina 578.00 2013-05-21 IT

7 7 Simon 632.80 2013-07-30 Operations

8 8 Guru 722.50 2014-06-17 Finance

# R - JSON Files

JSON file stores data as text in human-readable format. Json stands for JavaScript Object Notation. R can read JSON files using the rjson package.

## Install rjson Package

In the R console, you can issue the following command to install the rjson package.

install.packages("rjson")

## Input Data

Create a JSON file by copying the below data into a text editor like notepad. Save the file with a **.json** extension and choosing the file type as **all files(\*.\*)**.

{

"ID":["1","2","3","4","5","6","7","8" ],

"Name":["Rick","Dan","Michelle","Ryan","Gary","Nina","Simon","Guru" ],

"Salary":["623.3","515.2","611","729","843.25","578","632.8","722.5" ],

"StartDate":[ "1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013",

"7/30/2013","6/17/2014"],

"Dept":[ "IT","Operations","IT","HR","Finance","IT","Operations","Finance"]

}

## Read the JSON File

The JSON file is read by R using the function from **JSON()**. It is stored as a list in R.

# Load the package required to read JSON files.

library("rjson")

# Give the input file name to the function.

result <- fromJSON(file = "input.json")

# Print the result.

print(result)

When we execute the above code, it produces the following result −

$ID

[1] "1" "2" "3" "4" "5" "6" "7" "8"

$Name

[1] "Rick" "Dan" "Michelle" "Ryan" "Gary" "Nina" "Simon" "Guru"

$Salary

[1] "623.3" "515.2" "611" "729" "843.25" "578" "632.8" "722.5"

$StartDate

[1] "1/1/2012" "9/23/2013" "11/15/2014" "5/11/2014" "3/27/2015" "5/21/2013"

"7/30/2013" "6/17/2014"

$Dept

[1] "IT" "Operations" "IT" "HR" "Finance" "IT"

"Operations" "Finance"

## Convert JSON to a Data Frame

We can convert the extracted data above to a R data frame for further analysis using the **as.data.frame()** function.

# Load the package required to read JSON files.

library("rjson")

# Give the input file name to the function.

result <- fromJSON(file = "input.json")

# Convert JSON file to a data frame.

json\_data\_frame <- as.data.frame(result)

print(json\_data\_frame)

When we execute the above code, it produces the following result −

id, name, salary, start\_date, dept

1 1 Rick 623.30 2012-01-01 IT

2 2 Dan 515.20 2013-09-23 Operations

3 3 Michelle 611.00 2014-11-15 IT

4 4 Ryan 729.00 2014-05-11 HR

5 NA Gary 843.25 2015-03-27 Finance

6 6 Nina 578.00 2013-05-21 IT

7 7 Simon 632.80 2013-07-30 Operations

8 8 Guru 722.50 2014-06-17 Finance

**Working with R Charts and Graphs:-**

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

**Pie Chart:-**

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

### Syntax

The basic syntax for creating a pie-chart using the R is −

pie(x, labels, radius, main, col, clockwise)

Following is the description of the parameters used −

* **x** is a vector containing the numeric values used in the pie chart.
* **labels** is used to give description to the slices.
* **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
* **main** indicates the title of the chart.
* **col** indicates the color palette.
* **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

### Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

# Create data for the graph.

x <- c(21, 62, 10, 53)

labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.

png(file = "city.png")

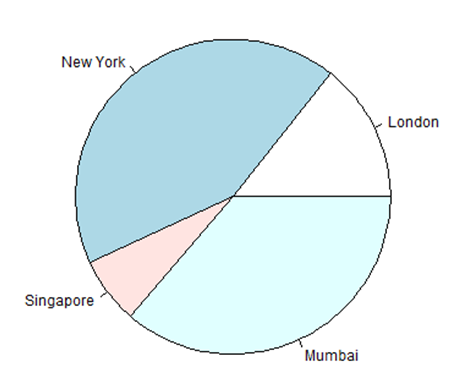
# Plot the chart.

pie(x,labels)

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



## Pie Chart Title and Colors

We can expand the features of the chart by adding more parameters to the function. We will use parameter **main** to add a title to the chart and another parameter is **col** which will make use of rainbow colour pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use length(x).

### Example

The below script will create and save the pie chart in the current R working directory.

# Create data for the graph.

x <- c(21, 62, 10, 53)

labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.

png(file = "city\_title\_colours.jpg")

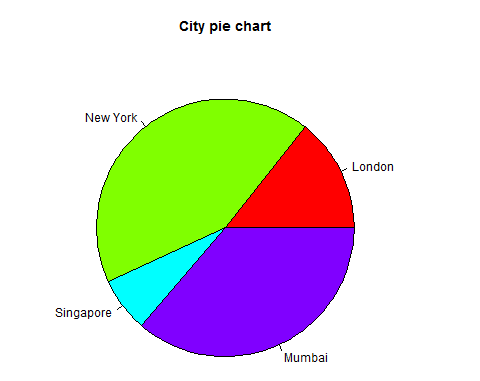
# Plot the chart with title and rainbow color pallet.

pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



## Slice Percentages and Chart Legend

We can add slice percentage and a chart legend by creating additional chart variables.

# Create data for the graph.

x <- c(21, 62, 10,53)

labels <- c("London","New York","Singapore","Mumbai")

piepercent<- round(100\*x/sum(x), 1)

# Give the chart file a name.

png(file = "city\_percentage\_legends.jpg")

# Plot the chart.

pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))

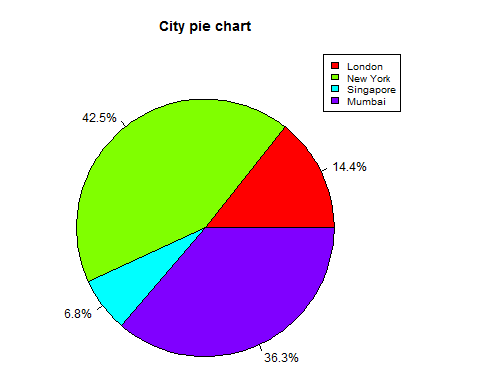
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,

fill = rainbow(length(x)))

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



## 3D Pie Chart

A pie chart with 3 dimensions can be drawn using additional packages. The package **plotrix** has a function called **pie3D()** that is used for this.

# Get the library.

library(plotrix)

# Create data for the graph.

x <- c(21, 62, 10,53)

lbl <- c("London","New York","Singapore","Mumbai")

# Give the chart file a name.

png(file = "3d\_pie\_chart.jpg")

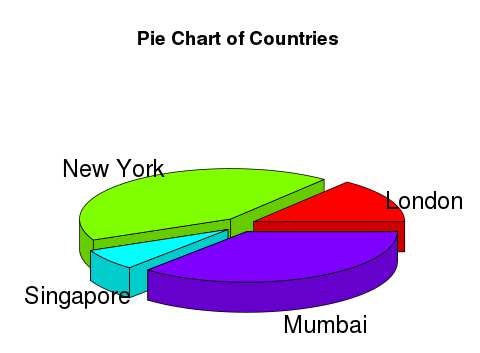
# Plot the chart.

pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



**Bar Chart:-**

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

### Syntax

The basic syntax to create a bar-chart in R is −

barplot(H,xlab,ylab,main, names.arg,col)

Following is the description of the parameters used −

* **H** is a vector or matrix containing numeric values used in bar chart.
* **xlab** is the label for x axis.
* **ylab** is the label for y axis.
* **main** is the title of the bar chart.
* **names.arg** is a vector of names appearing under each bar.
* **col** is used to give colors to the bars in the graph.

### Example

A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

# Create the data for the chart

H <- c(7,12,28,3,41)

# Give the chart file a name

png(file = "barchart.png")

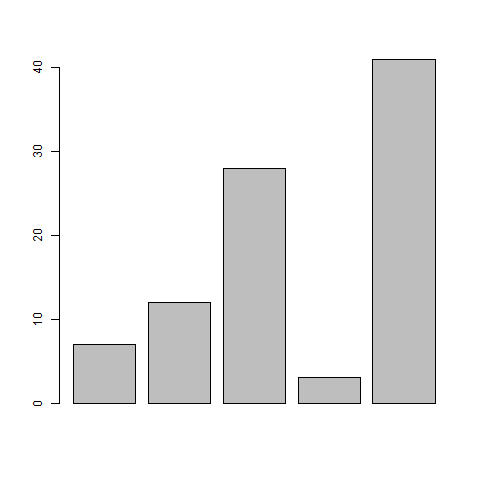
# Plot the bar chart

barplot(H)

# Save the file

dev.off()

When we execute above code, it produces following result −



## Bar Chart Labels, Title and Colors

The features of the bar chart can be expanded by adding more parameters. The **main** parameter is used to add **title**. The **col** parameter is used to add colors to the bars. The **args.name** is a vector having same number of values as the input vector to describe the meaning of each bar.

### Example

The below script will create and save the bar chart in the current R working directory.

# Create the data for the chart

H <- c(7,12,28,3,41)

M <- c("Mar","Apr","May","Jun","Jul")

# Give the chart file a name

png(file = "barchart\_months\_revenue.png")

# Plot the bar chart

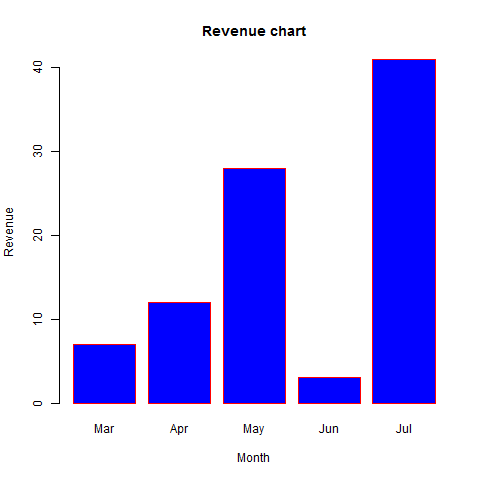
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",

main="Revenue chart",border="red")

# Save the file

dev.off()

When we execute above code, it produces following result −



## Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

# Create the input vectors.

colors = c("green","orange","brown")

months <- c("Mar","Apr","May","Jun","Jul")

regions <- c("East","West","North")

# Create the matrix of the values.

Values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow = 3, ncol = 5, byrow = TRUE)

# Give the chart file a name

png(file = "barchart\_stacked.png")

# Create the bar chart

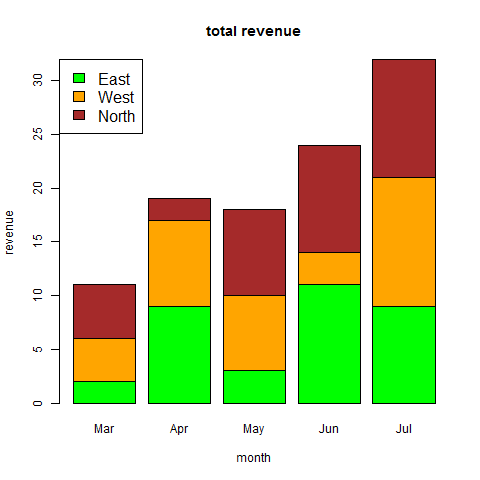
barplot(Values, main = "total revenue", names.arg = months, xlab = "month", ylab = "revenue", col = colors)

# Add the legend to the chart

legend("topleft", regions, cex = 1.3, fill = colors)

# Save the file

dev.off()



Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.

### Syntax

The basic syntax to create a boxplot in R is −

boxplot(x, data, notch, varwidth, names, main)

Following is the description of the parameters used −

* **x** is a vector or a formula.
* **data** is the data frame.
* **notch** is a logical value. Set as TRUE to draw a notch.
* **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
* **names** are the group labels which will be printed under each boxplot.
* **main** is used to give a title to the graph.

**Histogram:-**

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

### Syntax

The basic syntax for creating a histogram using R is −

hist(v,main,xlab,xlim,ylim,breaks,col,border)

Following is the description of the parameters used −

* **v** is a vector containing numeric values used in histogram.
* **main** indicates title of the chart.
* **col** is used to set color of the bars.
* **border** is used to set border color of each bar.
* **xlab** is used to give description of x-axis.
* **xlim** is used to specify the range of values on the x-axis.
* **ylim** is used to specify the range of values on the y-axis.
* **breaks** is used to mention the width of each bar.

### Example

A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.

# Create data for the graph.

v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.

png(file = "histogram.png")

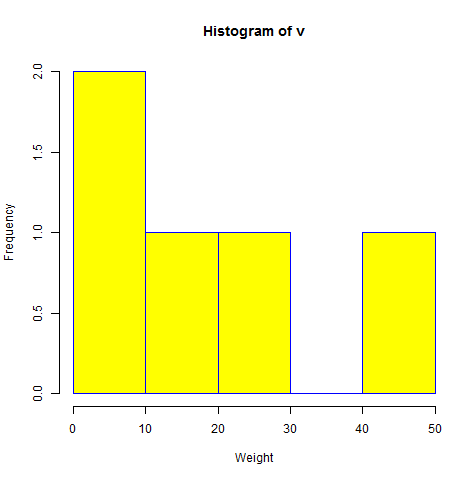
# Create the histogram.

hist(v,xlab = "Weight",col = "yellow",border = "blue")

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



## Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

The width of each of the bar can be decided by using breaks.

# Create data for the graph.

v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.

png(file = "histogram\_lim\_breaks.png")

# Create the histogram.

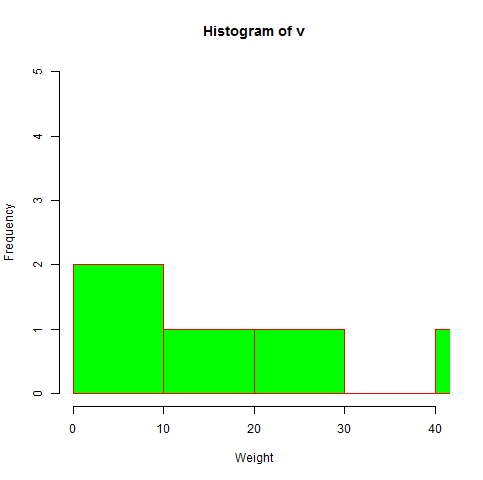
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),

breaks = 5)

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



**Line-graph:-**

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

### Syntax

The basic syntax to create a line chart in R is −

plot(v,type,col,xlab,ylab)

Following is the description of the parameters used −

* **v** is a vector containing the numeric values.
* **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
* **xlab** is the label for x axis.
* **ylab** is the label for y axis.
* **main** is the Title of the chart.
* **col** is used to give colors to both the points and lines.

### Example

A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory.

# Create the data for the chart.

v <- c(7,12,28,3,41)

# Give the chart file a name.

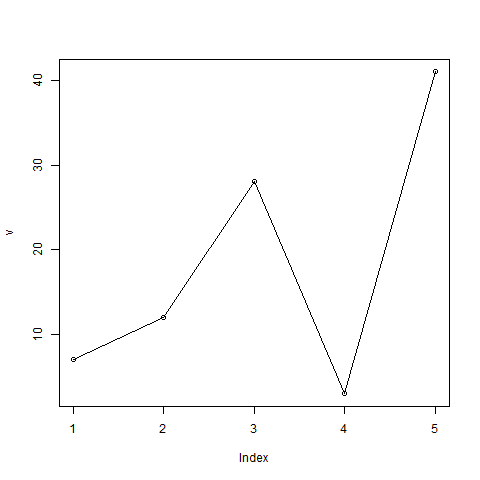
png(file = "line\_chart.jpg")

# Plot the bar chart.

plot(v,type = "o")

# Save the file.

dev.off()

When we execute the above code, it produces the following result −

## Line Chart Title, Color and Labels

The features of the line chart can be expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.

### Example

# Create the data for the chart.

v <- c(7,12,28,3,41)

# Give the chart file a name.

png(file = "line\_chart\_label\_colored.jpg")

# Plot the bar chart.

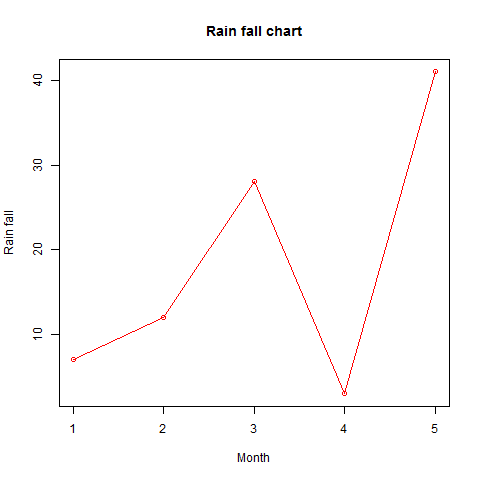
plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",

main = "Rain fall chart")

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



## Multiple Lines in a Line Chart

More than one line can be drawn on the same chart by using the **lines()**function.

After the first line is plotted, the lines() function can use an additional vector as input to draw the second line in the chart,

# Create the data for the chart.

v <- c(7,12,28,3,41)

t <- c(14,7,6,19,3)

# Give the chart file a name.

png(file = "line\_chart\_2\_lines.jpg")

# Plot the bar chart.

plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",

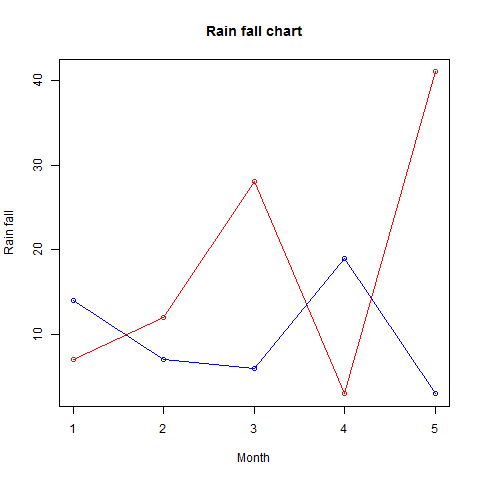
main = "Rain fall chart")

lines(t, type = "o", col = "blue")

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



**Scatter plots:-**

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

### Syntax

The basic syntax for creating scatterplot in R is −

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

Following is the description of the parameters used −

* **x** is the data set whose values are the horizontal coordinates.
* **y** is the data set whose values are the vertical coordinates.
* **main** is the tile of the graph.
* **xlab** is the label in the horizontal axis.
* **ylab** is the label in the vertical axis.
* **xlim** is the limits of the values of x used for plotting.
* **ylim** is the limits of the values of y used for plotting.
* **axes** indicates whether both axes should be drawn on the plot.

### Example

We use the data set **"mtcars"** available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

input <- mtcars[,c('wt','mpg')]

print(head(input))

When we execute the above code, it produces the following result −

wt mpg

Mazda RX4 2.620 21.0

Mazda RX4 Wag 2.875 21.0

Datsun 710 2.320 22.8

Hornet 4 Drive 3.215 21.4

Hornet Sportabout 3.440 18.7

Valiant 3.460 18.1

## Creating the Scatterplot

The below script will create a scatterplot graph for the relation between wt(weight) and mpg(miles per gallon).

# Get the input values.

input <- mtcars[,c('wt','mpg')]

# Give the chart file a name.

png(file = "scatterplot.png")

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.

plot(x = input$wt,y = input$mpg,

xlab = "Weight",

ylab = "Milage",

xlim = c(2.5,5),

ylim = c(15,30),

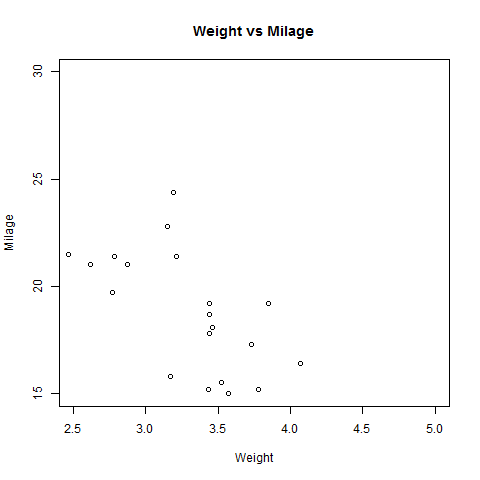
main = "Weight vs Milage"

)

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



## Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

### Syntax

The basic syntax for creating scatterplot matrices in R is −

pairs(formula, data)

Following is the description of the parameters used −

* **formula** represents the series of variables used in pairs.
* **data** represents the data set from which the variables will be taken.

### Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

# Give the chart file a name.

png(file = "scatterplot\_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~wt+mpg+disp+cyl,data = mtcars,

main = "Scatterplot Matrix")

# Save the file.

dev.off()

When the above code is executed we get the following output.

