**231CS33 PROGRAMMING FOR DATA SCIENCE**

## UNIT – I INTRODUCTION

**Data Science: Introduction to Data Science – Digital Universe – Sources of Data – Information Commons – Data Science Project Life Cycle**.

### 1.1. Data Science:

Data science is the domain of study that deals with vast volumes of data using modern tools and techniques, including essential data science skills, to find unseen patterns, derive meaningful information, and make business decisions. Data science uses complex machine learning algorithms to build predictive models. The data used for analysis can come from many different sources and presented in various formats. Now that you know what data science is, let's see the data science lifestyle.

**Uses of Data Science:**

1. Data science may detect patterns in seemingly unstructured or unconnected data, allowing conclusions and predictions to be made.
2. Tech businesses that acquire user data can utilise strategies to transform that data into valuable or profitable information.
3. Data Science has also made inroads into the transportation industry, such as with driverless cars. It is simple to lower the number of accidents with the use of driverless cars. For example, with driverless cars, training data is supplied to the algorithm, and the data is examined using data Science approaches, such as the speed limit on the highway, busy streets, etc.
4. Data Science applications provide a better level of therapeutic customisation through genetics and genomics research.

**Applications of Data Science**

There are various applications of data science, including:

### 1. Healthcare

Healthcare companies are using data science to build sophisticated medical instruments to detect and cure diseases.

### 2. Gaming

Video and computer games are now being created with the help of data science and that has taken the gaming experience to the next level.

**3. Image Recognition**

Identifying patterns is one of the most commonly known applications of data science. in images and detecting objects in an image is one of the most popular data science applications.

**4. Recommendation Systems**

Next up in the data science and its applications list comes Recommendation Systems. Netflix and Amazon give movie and product recommendations based on what you like to watch, purchase, or browse on their platforms.

**5. Logistics**

Data Science is used by logistics companies to optimize routes to ensure faster delivery of products and increase operational efficiency.

**6. Fraud Detection**

Fraud detection comes the next in the list of applications of data science. Banking and financial institutions use data science and related algorithms to detect fraudulent transactions.

**7. Internet Search**

Internet comes the next in the list of applications of data science. When we think of search, we immediately think of Google. Right? However, there are other search engines, such as Yahoo, Duckduckgo, Bing, AOL, Ask, and others, that employ data science algorithms to offer the best results for our searched query in a matter of seconds. Given that Google handles more than 20 petabytes of data per day. Google would not be the 'Google' we know today if data science did not exist.

**8. Speech recognition**

Speech recognition is one of the most commonly known applications of data science. It is a technology that enables a computer to recognize and transcribe spoken language into text. It has a wide range of applications, from virtual assistants and voice-controlled devices to automated customer service systems and transcription services.

**9. Targeted Advertising**

If you thought Search was the most essential data science use, consider this: the whole digital marketing spectrum. From display banners on various websites to digital billboards at airports, data science algorithms are utilised to identify almost anything. This is why digital advertisements have a far higher CTR (Call-Through Rate) than traditional marketing. They can be customised based on a user's prior behaviour. That is why you may see adverts for Data Science Training Programs while another person sees an advertisement for clothes in the same region at the same time.

### 10. Airline Route Planning

Next up in the data science and its applications list comes route planning. As a result of data science, it is easier to predict flight delays for the airline industry, which is helping it grow. It also helps to determine whether to land immediately at the destination or to make a stop in between, such as a flight from Delhi to the United States of America or to stop in between and then arrive at the destination.

### 11. Augmented Reality

Last but not least, the final data science applications appear to be the most fascinating in the future. Yes, we are discussing something other than augmented reality. Do you realise there's a fascinating relationship between data science and virtual reality? A virtual reality headset incorporates computer expertise, algorithms, and data to create the greatest viewing experience possible. The popular game Pokemon GO is a minor step in that direction. The ability to wander about and look at Pokemon on walls, streets, and other non-existent surfaces. The makers of this game chose the locations of the Pokemon and gyms using data from Ingress, the previous app from the same business.

#### Example of Data Science

Here are some brief examples of data science showing data science's versatility.

- **Law Enforcement:** In this scenario, data science is used to help police in Belgium to better understand where and when to deploy personnel to prevent crime. With only limited resources and a large area to cover data science used dashboards and reports to increase the officers' situational awareness, allowing a police force that's spread thin to maintain order and anticipate criminal activity.
- **Pandemic Fighting:** The state of Rhode Island wanted to reopen schools, but was naturally cautious, considering the ongoing COVID-19 pandemic. The state used data science to expedite case investigations and contact tracing, enabling a small staff to handle an overwhelming number of concerned calls from citizens. This information helped the state set up a call center and coordinate preventative measures.
- **Driverless Vehicles:** Lunewave, a sensor manufacturing company, was looking for a way to make sensor technology more cost-effective and accurate. They turned to data science and machine learning to train their sensors to be safer and more reliable, as well as using data to improve their 3D-printed sensor manufacturing process.

#### The Data Science Lifecycle

Data science's lifecycle consists of five distinct stages, each with its own tasks:

1. **Capture:** Data Acquisition, Data Entry, Signal Reception, Data Extraction. This stage involves gathering raw structured and unstructured data.
2. **Maintain:** Data Warehousing, Data Cleansing, Data Staging, Data Processing, and Data Architecture. This stage covers taking the raw data and putting it in a form that can be used.
3. **Process:** Data Mining, Clustering/Classification, Data Modeling, Data Summarization. Data scientists take the prepared data and examine its patterns, ranges, and biases to determine how useful it will be in predictive analysis.

4. **Analyze:** Exploratory/Confirmatory, Predictive Analysis, Regression, Text Mining, and Qualitative Analysis. Here is the real meat of the lifecycle. This stage involves performing the various analyses on the data.
5. **Communicate:** Data Reporting, Data Visualization, Business Intelligence, and Decision Making. In this final step, analysts prepare the analyses in easily readable forms such as charts, graphs, and reports.

**Who Oversees the Data Science Process?**

**1. Business Managers:** The business managers are the people in charge of overseeing the data science training method. Their primary responsibility is to collaborate with the data science team to characterise the problem and establish an analytical method. A data scientist may oversee the marketing, finance, or sales department, and report to an executive in charge of the department. Their goal is to ensure projects are completed on time by collaborating closely with data scientists and IT managers.
**2. IT Managers:** Following them are the IT managers. If the member has been with the organisation for a long time, the responsibilities will undoubtedly be more important than any others. They are primarily responsible for developing the infrastructure and architecture to enable data science activities. Data science teams are constantly monitored and resourced accordingly to ensure that they operate efficiently and safely. They may also be in charge of creating and maintaining IT environments for data science teams.
**3. Data Science Managers:** The data science managers make up the final section of the tea. They primarily trace and supervise the working procedures of all data science team members. They also manage and keep track of the day-to-day activities of the three data science teams. They are team builders who can blend project planning and monitoring with team growth.

**What is a Data Scientist?**

If learning what is data science sounded interesting, understanding what does this job roles is all about will me much more interesting to you. Data scientists are among the most recent analytical data professionals who have the technical ability to handle complicated issues as well as the desire to investigate what questions need to be answered. They're a mix of mathematicians, computer scientists, and trend forecasters. They're also in high demand and well-paid because they work in both the business and IT sectors. On a daily basis, a data scientist may do the following tasks:

1. Discover patterns and trends in datasets to get insights
2. Create forecasting algorithms and data models
3. Improve the quality of data or product offerings by utilising machine learning techniques
4. Distribute suggestions to other teams and top management
5. In data analysis, use data tools such as R, SAS, Python, or SQL
6. Top the field of data science innovations

**What Does a Data Scientist Do?**

Data scientist solves business problems through a series of steps, including:

- Before tackling the data collection and analysis, the data scientist determines the problem by asking the right questions and gaining understanding.
- The data scientist then determines the correct set of variables and data sets.
- The data scientist gathers structured and unstructured data from many disparate sources—enterprise data, public data, etc.
- Once the data is collected, the data scientist processes the raw data and converts it into a format suitable for analysis. This involves cleaning and validating the data to guarantee uniformity, completeness, and accuracy.
- After the data has been rendered into a usable form, it's fed into the analytic system—ML algorithm or a statistical model. This is where the data scientists analyze and identify patterns and trends.
- When the data has been completely rendered, the data scientist interprets the data to find opportunities and solutions.
- The data scientists finish the task by preparing the results and insights to share with the appropriate stakeholders and communicating the results.

### 1.2. Digital Universe:

The term "digital universe" in data science refers to the entirety of digital data generated, stored, processed, and transmitted globally. This encompasses all types of data created by various sources such as individuals, organizations, and devices. The concept is significant in understanding the scope, scale, and impact of digital information on various aspects of society, economy, and technology.

Here are key components and aspects of the digital universe in data science:

1. **Volume**: The digital universe is characterized by its vast and exponentially growing volume of data. This includes data from social media, financial transactions, IoT devices, multimedia content, and more.
2. **Variety**: Data in the digital universe comes in multiple formats and types, including structured data (databases, spreadsheets), unstructured data (text, images, videos), and semi-structured data (JSON, XML).
3. **Velocity**: The speed at which data is generated and processed is a critical aspect. Real-time data from sensors, social media feeds, and financial markets exemplifies high-velocity data.
4. **Veracity**: Ensuring the accuracy and reliability of data is crucial. The digital universe contains both high-quality data and significant amounts of noise and misinformation.
5. **Value**: The ultimate goal of analyzing the digital universe is to extract valuable insights that can inform decision-making, drive innovation, and create competitive advantages.
6. **Big Data Technologies**: The digital universe necessitates the use of advanced technologies and tools for storage, processing, and analysis. These include distributed computing frameworks (Hadoop, Spark), data warehouses, cloud storage solutions, and machine learning algorithms.
7. **Data Sources**: The digital universe is populated by a diverse array of data sources, including social media platforms, sensors (IoT), mobile devices, enterprise systems, scientific research, and government databases.
8. **Privacy and Security**: Managing the digital universe involves addressing significant challenges related to data privacy, security, and compliance with regulations like GDPR and CCPA.

9. **Impact on Society**: The digital universe influences many aspects of modern life, including business operations, healthcare, education, entertainment, and public policy. It drives innovations in artificial intelligence, machine learning, and other cutting-edge technologies.
10. **Sustainability**: The rapid growth of the digital universe also raises concerns about the environmental impact of data centers and the need for sustainable practices in managing digital data.

**Uses of Digital Universe:**

The concept of the digital universe is essential in data science due to its expansive and diverse data landscape, which provides numerous opportunities for analysis, insights, and decision-making across various fields. Here are the key uses of the digital universe in data science:

**1. Insight Generation**

Data scientists analyze vast amounts of data from the digital universe to uncover patterns, trends, and correlations. This can lead to valuable insights for businesses, healthcare, finance, and more. For instance, analyzing social media data can reveal consumer sentiment and market trends.

**2. Predictive Analytics**

Predictive models built on data from the digital universe help forecast future events or behaviors. Retailers, for example, use predictive analytics to anticipate inventory needs based on past sales data, while healthcare providers predict patient outcomes from medical records.

**3. Personalization**

Data from the digital universe allows for highly personalized experiences. Online services like streaming platforms, e-commerce websites, and social media use data science to tailor recommendations and advertisements to individual user preferences and behaviors.

**4. Operational Efficiency**

Organizations leverage data from the digital universe to streamline operations and improve efficiency. Manufacturing processes can be optimized using IoT data, while logistics companies use real-time data for route optimization and fleet management.

**5. Fraud Detection and Security**

Financial institutions and online platforms use data science to detect fraudulent activities by analyzing transaction patterns and user behavior. Security systems can also monitor and analyze data traffic to identify and mitigate cyber threats.

**6. Healthcare Advancements**

The digital universe provides critical data for medical research and patient care. Analyzing electronic health records, medical imaging, and genomic data helps in disease diagnosis, treatment planning, and personalized medicine.

**7. Scientific Research**

Researchers across various fields use data from the digital universe to advance scientific knowledge. Environmental scientists, for example, analyze satellite data to monitor climate change, while astronomers use data from telescopes to study the universe.

**8. Business Intelligence**

Businesses use data from the digital universe for strategic planning and decision-making. Data science tools analyze sales data, customer feedback, and market trends to guide business strategies and improve competitiveness.

**9. Urban Planning and Smart Cities**

Data from sensors, mobile devices, and public services in the digital universe is used to enhance urban planning and develop smart cities. This includes traffic management, energy distribution, waste management, and public safety.

**10. Educational Insights**

Educational institutions analyze data from the digital universe to improve learning outcomes. This includes student performance data, engagement metrics, and feedback to personalize education and identify areas for improvement.

**11. Economic and Social Policy**

Governments and organizations use data science to inform policy decisions. Economic data analysis helps in understanding market dynamics and labor trends, while social data analysis aids in addressing public health issues and resource allocation.

**12. Sustainability and Environmental Monitoring**

Data from the digital universe helps monitor and manage environmental resources. This includes tracking deforestation, pollution levels, and wildlife populations to promote sustainable practices and environmental conservation.

**Aspects of the Digital Universe in Data Science**

1. **Big Data Technologies**

- **Tools and Frameworks**: Technologies such as Hadoop, Spark, and NoSQL databases enable the processing and analysis of large datasets.

- **Cloud Computing**: Cloud platforms like AWS, Google Cloud, and Azure provide scalable resources for managing big data workloads.

2. **Data Analytics**

- **Descriptive Analytics**: Summarizing historical data to understand what has happened.
- **Predictive Analytics:** Using statistical models and machine learning to predict future outcomes.
- **Prescriptive Analytics:** Recommending actions based on data insights to achieve desired outcomes.

3. **Machine Learning and AI**

- **Algorithms**: Applying machine learning algorithms to analyze patterns, classify data, and make predictions.
- **Deep Learning**: Leveraging neural networks for complex tasks such as image recognition, natural language processing, and autonomous systems.

4. **Data Governance**

- **Policies and Standards:** Establishing frameworks for data management, privacy, and security.
- **Compliance:** Adhering to regulations like GDPR, CCPA, and industry-specific standards.

5. **Data Visualization**

- **Techniques**: Creating visual representations of data to make insights more accessible and understandable.
- **Tools**: Utilizing tools like Tableau, Power BI, and D3.js to develop interactive and informative visualizations.

6. **Impact on Society**

- **Economic:** Driving innovation, improving efficiency, and creating new business opportunities.
- **Social:** Enhancing quality of life through smart cities, personalized healthcare, and improved education.
- **Ethical:** Addressing concerns related to data privacy, security, and the ethical use of data.

### 1.3. Sources of Data

A data source may be the initial location where data is born or where physical information is first digitized, however even the most refined data may serve as a source, as long as another process accesses and utilizes it. Concretely, a data source may be a database, a flat file, live

measurements from physical devices, scraped web data, or any of the myriad static and streaming data services which abound across the internet.

Here's an example of a data source in action. Imagine a fashion brand selling products online. To display whether an item is out of stock, the website gets information from an inventory database. In this case, the inventory tables are a data source, accessed by the web application which serves the website to customers.

Focusing on how the term is used in the familiar database management context will help to clarify what kinds of data sources exist, how they work, and when they are useful.

**Data source nomenclature:**

Databases remain the most common data sources, as the primary stores for data in ubiquitous relational database management systems (RDBMS). In this context, an important concept is the Data Source Name (DSN). The DSN is defined within destination databases or applications as a pointer to the actual data, whether it exists locally or is found on a remote server (and whether in a single physical location or virtualized.) The DSN is not necessarily the same as the relevant database name or file name, rather it is in an address or label used to easily reach the data at its source.

Ultimately, the systems doing the ingesting (of data) determine the context for any discussion around data sources, so definitions and nomenclature vary widely and may be confusing. This is especially true in more technical documentation. For example, within the Java software platform, a 'Datasource' refers specifically to an object representing a connection to a database (like an extensible, programmatically packaged DSN). Meanwhile, some newer platforms use 'DataSource' more widely to mean any collection of data which provides a standardized means for access.

**Data source types**

Though the diversity of content, format, and location for data is only increasing with contributions from technologies such as IoT and the adoption of big data methodologies, it remains possible to classify most data sources into two broad categories: machine data sources and file date sources.

Though both share the same basic purpose — pointing to the data's location and describing similar connection characteristics — machine and file data sources are stored, accessed, and used in different ways.

**Machine data sources**

Machine data sources have names defined by users, must reside on the machine that is ingesting data, and cannot be easily shared. Like other data sources, machine data sources provide all the information necessary to connect to data, such as relevant software drivers and

a driver manager, but users need only ever refer to the DSN as shorthand to invoke the connection or query the data.

The connection information is stored in environment variables, database configuration options, or a location internal to the machine or application being used. An Oracle data source, for example, will contain a server location for accessing the remote DBMS, information about which drivers to use, the driver engine, and any other relevant parts of a typical connection string, such as system and user IDs and authentication.

**File data sources**

File data sources contain all of the connection information inside a single, shareable, computer file (typically with a .dsn extension). Users do not decide which name is assigned to file data sources, as these sources are not registered to individual applications, systems, or users, and in fact do not have a DSN like that of machine data sources. Each file stores a connection string for a single data source.

File data sources, unlike machine sources, are editable and copyable like any other computer file. This allows users and systems to share a common connection (by moving the data source between individual machines or servers), and for the streamlining of data connection processes (for example by keeping a source file on a shared resource so it may be used simultaneously by multiple applications and users).

It is important to note that 'unshareable' .dsn files also exist. These are the same type of file as described above, but they exist on a single machine and cannot be moved or copied. These files point directly to machine data sources. This means that unshareable file data sources are wrappers for machine data sources, serving as a proxy for applications which expect only files but also need to connect to machine data.

**How data sources work**

Data sources are used in a variety of ways. Data can be transported thanks to diverse network protocols, such as the well-known File Transfer Protocol (FTP) and HyperText Transfer Protocol (HTTP), or any of the myriad Application Programming Interfaces (APIs) provided by websites, networked applications, and other services.

Many platforms use data sources with FTP addresses to specify the location of data needed to be imported. For example, in the Adobe Analytics platform, a file data source is uploaded to a server using an FTP client, then a service utilizes this source to move and process the relevant data automatically.

SFTP (The S stands for Secure or SSH) is used when usernames and passwords need to be obfuscated and content encrypted, or FTPS may alternatively be used by adding Transport Layer Security (TLS) to FTP, achieving the same goal.

Meanwhile, many and diverse APIs are now provided to manage data sources and how they are used in applications. APIs are used to programmatically link applications to data sources, and typically provide more customization and a more versatile collection of access methods. For example, Spark provides an API with abstract implementations for representing and connecting to data sources, from barebones but extensible classes for generic relational sources, to detailed implementations for hard-coded JDBC connections.

Other protocols for moving data from sources to destinations, especially on the web, include NFS, SMB, SOAP, REST, and WebDAV. These protocols are often used within APIs (and some APIs themselves make use of other APIs internally), within fully featured data applications, or as standalone transfer processes. Each have characteristic features and security concerns which should be considered for any data transfer.

**The purpose of a data source**

Ultimately, data sources are intended to help users and applications connect to and move data to where it needs to be. They gather relevant technical information in one place and hide it so data consumers can focus on processing and identify how to best utilize their data.

The purpose here is to package connection information in a more easily understood and user-friendly format. This makes data sources critical for more easily integrating disparate systems, as they save shareholders from the need to deal with and troubleshoot complex but low-level connection information.

And although this connection information is hidden, it is always accessible when necessary. Additionally, this information is stored in consistent locations and formats which can ease other processes such as migrations or planned system structural changes.

### 1.4. Information Commons:

"Information Commons" refers to shared resources and collaborative environments that facilitate access to and use of information and data for research, education, and innovation. Information Commons are designed to provide the infrastructure and support needed for data-driven work, fostering a culture of openness, collaboration, and knowledge sharing.

**Key Components of an Information Commons in Data Science**

1. **Data Repositories**

   - Centralized databases where datasets are stored, cataloged, and made accessible to researchers and practitioners.
   - Examples include public datasets like those from the U.S. Census Bureau, scientific datasets like those from the Human Genome Project, or open data portals from city governments.

2. **Computing Resources**

- High-performance computing (HPC) facilities, cloud computing services, and data storage solutions that provide the necessary computational power and storage capacity for data-intensive tasks.
- Resources such as AWS, Google Cloud Platform, Microsoft Azure, or institution-specific HPC cluster.

3. **Collaborative Workspaces**

- Physical or virtual spaces designed for collaborative work, often equipped with advanced tools and technologies to support data analysis, visualization, and research collaboration.
- Examples include university-based data science labs, coworking spaces for tech startups, and virtual collaboration platforms like Jupyter Notebooks and GitHub.

4. **Software and Tools**

- Access to a variety of software tools and platforms essential for data science work, such as statistical analysis software (R, SAS), machine learning frameworks (TensorFlow, PyTorch), and data visualization tools (Tableau, D3.js).
- Licensing agreements and training sessions to facilitate the use of these tools.

5. **Knowledge Repositories**

- Libraries and digital archives that provide access to research papers, technical reports, case studies, and other scholarly resources.
- Examples include digital libraries like IEEE Xplore, arXiv, PubMed, and institutional repositories.

6. **Educational and Training Programs**

- Workshops, courses, seminars, and online tutorials aimed at building skills in data science, programming, statistical analysis, machine learning, and other relevant areas.
- Partnerships with educational platforms like Coursera, edX, DataCamp, and in-house training programs.

7. **Data Governance and Ethics Frameworks**

- Policies and guidelines to ensure the responsible use and sharing of data, addressing issues such as data privacy, security, intellectual property, and ethical considerations.
- Compliance with regulations like GDPR, CCPA, and institutional review boards (IRBs).

8. **Community and Networking**

- Support for building and maintaining a community of data scientists, researchers, students, and professionals through events, conferences, hackathons, and online forums.

- Examples include meetups, professional associations like the Data Science Association, and online communities such as Kaggle and Stack Overflow.

**Benefits of Information Commons in Data Science**

1. **Enhanced Collaboration**

By providing shared resources and collaborative spaces, Information Commons foster interdisciplinary collaboration and collective problem-solving.

2. **Increased Access to Data and Tools**

Making data and computational tools widely accessible democratizes the ability to perform advanced data science tasks, enabling more people to contribute to scientific and technological advancements.

3. **Accelerated Research and Innovation**

Access to high-quality data and computational resources speeds up the research process, leading to faster innovation and discovery.

4. **Skill Development**

Educational programs and resources help individuals build and enhance their data science skills, contributing to a more knowledgeable and skilled workforce.

5. **Promotion of Open Science**

Information Commons support the principles of open science by promoting transparency, reproducibility, and the sharing of data and findings.

6. **Support for Data-Driven Decision Making**

By providing robust data and analytical tools, Information Commons help organizations make more informed, evidence-based decisions.

## 1.5.Data Science Project Life Cycle

Data Science Lifecycle revolves around the use of machine learning and different analytical strategies to produce insights and predictions from information in order to acquire a commercial enterprise objective. The complete method includes a number of steps like data cleaning, preparation, modelling, model evaluation, etc. It is a lengthy procedure and may additionally take quite a few months to complete. So, it is very essential to have a generic structure to observe for each and every hassle at hand. The globally mentioned structure in fixing any analytical problem is referred to as a Cross Industry Standard Process for Data Mining or CRISP-DM framework.

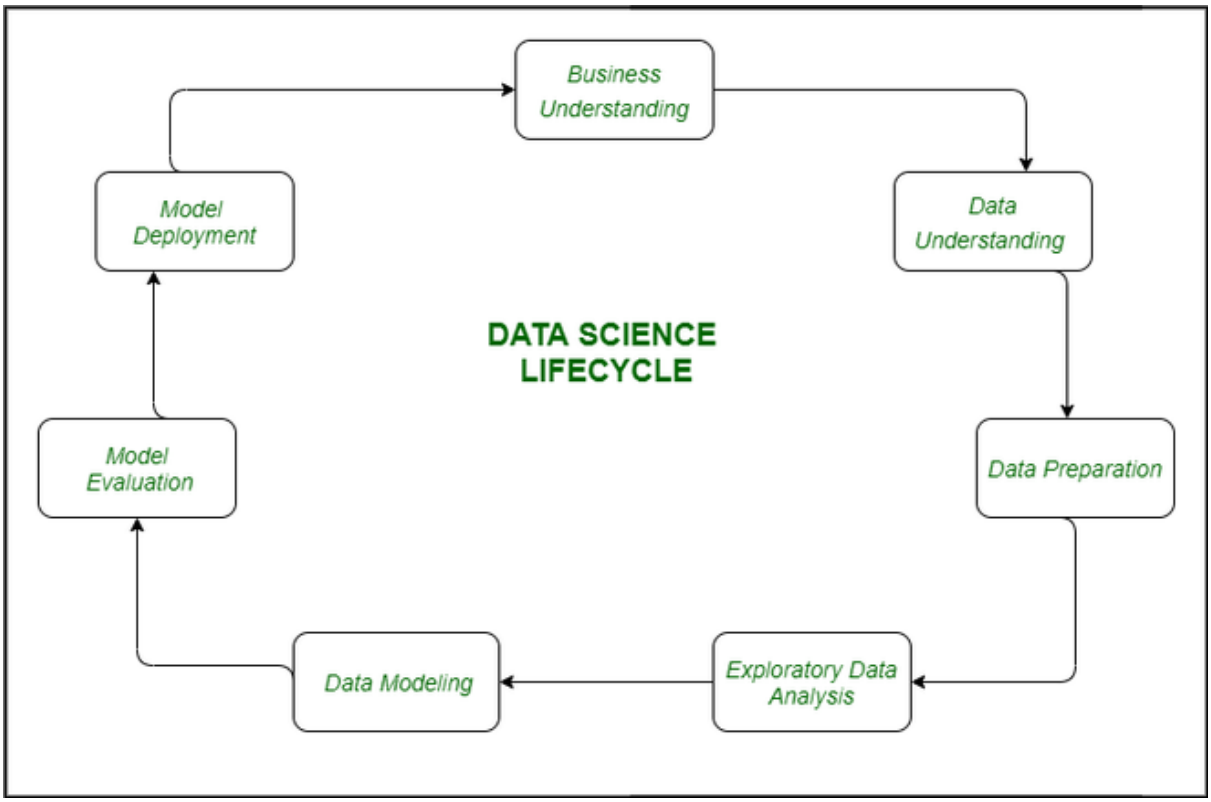
**What is the need for Data Science?**

Earlier data used to be much less and generally accessible in a well-structured form, that we could save effortlessly and easily in Excel sheets, and with the help of Business Intelligence tools data can be processed efficiently. But Today we used to deals with large amounts of data like about 3.0 quintals bytes of records is producing on each and every day, which ultimately results in an explosion of records and data. According to recent researches, It is estimated that 1.9 MB of data and records are created in a second that too through a single individual.

So this very big challenge for any organization to deal with such a massive amount of data generating every second. For handling and evaluating this data we required some very powerful, complex algorithms and technologies and this is where Data science comes into the picture.

The following are some primary motives for the use of Data science technology:

1. It helps to convert the big quantity of uncooked and unstructured records into significant insights.
2. It can assist in unique predictions such as a range of surveys, elections, etc.
3. It also helps in automating transportation such as growing a self-driving car; we can say which the future of transportation is.
4. Companies are shifting towards Data science and opting for this technology. Amazon, Netflix, etc, which cope with the big quantity of data, are the use of information science algorithms for higher consumer experience.




**The lifecycle of Data Science**

**1. Business Understanding:** The complete cycle revolves around the enterprise goal. What will you resolve if you do not longer have a specific problem? It is extraordinarily essential to apprehend the commercial enterprise goal sincerely due to the fact that will be your ultimate aim of the analysis. After desirable perception only we can set the precise aim of evaluation that is in sync with the enterprise objective. You need to understand if the customer desires to minimize savings loss, or if they prefer to predict the rate of a commodity, etc.

**2. Data Understanding:** After enterprise understanding, the subsequent step is data understanding. This includes a series of all the reachable data. Here you need to intently work with the commercial enterprise group as they are certainly conscious of what information is present, what facts should be used for this commercial enterprise problem, and different information. This step includes describing the data, their structure, their relevance, their records type. Explore the information using graphical plots. Basically, extracting any data that you can get about the information through simply exploring the data.

**3. Preparation of Data:** This consists of steps like choosing the applicable data, integrating the data by means of merging the data sets, cleaning it, treating the lacking values through either eliminating them or imputing them, treating inaccurate data through eliminating them, additionally test for outliers the use of box plots and cope with them. Constructing new data, derive new elements from present ones. Format the data into the preferred structure, eliminate undesirable columns and features. Data preparation is the most time-consuming but arguably the most essential step in the complete existence cycle. Your model will be as accurate as your data.

**4. Exploratory Data Analysis:** This step includes getting some concept about the answer and elements affecting it, earlier than constructing the real model. Distribution of data inside distinctive variables of a character is explored graphically the usage of bar-graphs, Relations between distinct aspects are captured via graphical representations like scatter plots and warmth maps. Many data visualization strategies are considerably used to discover each and every characteristic individually and by means of combining them with different features.

**5. Data Modeling:** Data modeling is the coronary heart of data analysis. A model takes the organized data as input and gives the preferred output. This step consists of selecting the suitable kind of model, whether the problem is a classification problem, or a regression problem or a clustering problem. After deciding on the model family, amongst the number of algorithms amongst that family, we need to cautiously pick out the algorithms to put into effect and enforce them. We need to tune the hyperparameters of every model to obtain the preferred performance. We additionally need to make positive there is the right stability between overall performance and generalizability. We do no longer desire the model to study the data and operate poorly on new data.

**6. Model Evaluation:** Here the model is evaluated for checking if it is geared up to be deployed. The model is examined on an unseen data, evaluated on a cautiously thought out set of assessment metrics. We additionally need to make positive that the model conforms to reality. If we do not acquire a quality end result in the evaluation, we have to re-iterate the complete modelling procedure until the preferred stage of metrics is achieved. Any data science solution, a machine learning model, simply like a human, must evolve, must be capable to enhance itself with new data, adapt to a new evaluation metric. We can construct more than one model for a certain phenomenon, however, a lot of them may additionally be imperfect. The model assessment helps us select and construct an ideal model.

**7. Model Deployment:** The model after a rigorous assessment is at the end deployed in the preferred structure and channel. This is the last step in the data science life cycle. Each step in the data science life cycle defined above must be laboured upon carefully. If any step is performed improperly, and hence, have an effect on the subsequent step and the complete effort goes to waste. For example, if data is no longer accumulated properly, you'll lose records and you will no longer be constructing an ideal model. If information is not cleaned properly, the model will no longer work. If the model is not evaluated properly, it will fail in the actual world. Right from Business perception to model deployment, every step has to be given appropriate attention, time, and effort.

## UNIT – II

## DATA PREPROCESSING

Introduction to Data Preprocessing – Reading, Selecting, Filtering Data – Filtering Missing Values– Manipulating, Sorting, Grouping, Rearranging, Ranking Data.

### 2.1. Data Pre-processing:

Data Preprocessing can be defined as a process of converting raw data into a format that is understandable and usable for further analysis. It is an important step in the Data Preparation stage. It ensures that the outcome of the analysis is **accurate**, **complete**, and **consistent**.

### Understanding Data

The main objective of Data Understanding is to gather general insights about the input dataset that will help to perform further steps to preprocess data. Let's review two of the most common ways to understand input datasets.

#### Data Types

Data Type can be defined as labeling the values a feature can hold. The data type will also determine what kinds of relational, mathematical, or logical operations can be performed on it. A few of the most common data types include Integer, Floating, Character, String, Boolean, Array, Date, Time, etc.

#### Data Summary

Data Summary can be defined as generating descriptive or summary statistics for the features in a given dataset. For example, for a numeric column, it will compute mean, max, min, std, etc. For a categorical variable, it will compute the count of unique labels, labels with the highest frequency, etc.

### What is Data Pre-processing

Real-world datasets are generally messy, raw, incomplete, inconsistent, and unusable. It can contain manual entry errors, missing values, inconsistent schema, etc. Data Pre-processing is the process of converting raw data into a format that is understandable and usable. It is a crucial step in any Data Science project to carry out an efficient and accurate analysis. It ensures that data quality is consistent before applying any **Machine Learning** or **Data Mining techniques**.

### Why is Data Pre-processing Important

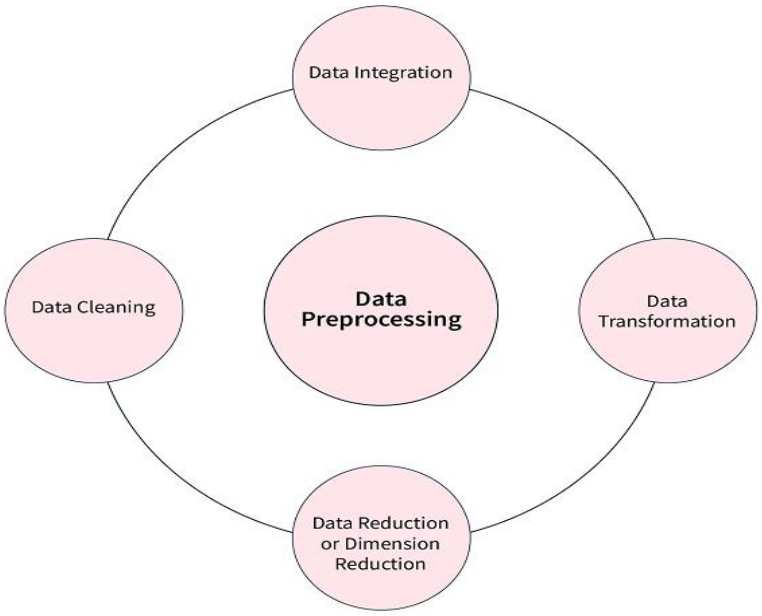
Data Pre-processing is an important step in the Data Preparation stage of a Data Science development lifecycle that will ensure reliable, robust, and consistent results. The main objective of this step is to ensure and check the quality of data before applying any Machine Learning or Data Mining methods. Let's review some of its benefits -

- **Accuracy** - Data Pre-processing will ensure that input data is accurate and reliable by ensuring there are no manual entry errors, no duplicates, etc.
- **Completeness** - It ensures that missing values are handled, and data is complete for further analysis.
- **Consistent** - Data Pre-processing ensures that input data is consistent, i.e., the same data kept in different places should match.
- **Timeliness** - Whether data is updated regularly and on a timely basis or not.
- **Trustable** - Whether data is coming from trustworthy sources or not.
- **Interpretability** - Raw data is generally unusable, and Data Pre-processing converts raw data into an interpretable format.

**Key Steps in Data Pre-processing**

Let's explore a few of the key steps involved in the Data Pre-processing stage –

Steps involved in the Data Preprocessing stage




**Data Cleaning**

Data Cleaning uses methods to handle incorrect, incomplete, inconsistent, or missing values. Some of the techniques for Data Cleaning include -

**Handling Missing Values**

- Input data can contain missing or **NULL** values, which must be handled before applying any Machine Learning or Data Mining techniques.
- Missing values can be handled by many techniques, such as removing rows/columns containing NULL values and imputing NULL values using mean, mode, regression, etc.

**De-noising**

- De-noising is a process of removing noise from the data. Noisy data is meaningless data that is not interpretable or understandable by machines or humans. It can occur due to data entry errors, faulty data collection, etc.
- De-noising can be performed by applying many techniques, such as binning the features, using regression to smoothen the features to reduce noise, clustering to detect the outliers, etc.

**Data Integration**

Data Integration can be defined as combining data from multiple sources. A few of the issues to be considered during Data Integration include the following -

- **Entity Identification Problem** - It can be defined as identifying objects/features from multiple databases that correspond to the same entity. For example, in database A _customer_id,_ and in database B _customer_number_ belong to the same entity.
- **Schema Integration** - It is used to merge two or more database schema/metadata into a single schema. It essentially takes two or more schema as input and determines a mapping between them. For example, entity type CUSTOMER in one schema may have CLIENT in another schema.
- **Detecting and Resolving Data Value Concepts** - The data can be stored in various ways in different databases, and it needs to be taken care of while integrating them into a single dataset. For example, dates can be stored in various formats such as **DD/MM/YYYY**, **YYYY/MM/DD**, or **MM/DD/YYYY**, etc.

**Data Reduction**

- **Data Reduction** is used to reduce the volume or size of the input data. Its main objective is to reduce storage and analysis costs and improve storage efficiency. A few of the popular techniques to perform Data Reduction include -
- **Dimensionality Reduction** - It is the process of reducing the number of features in the input dataset. It can be performed in various ways, such as selecting features with the highest importance, Principal Component Analysis (PCA), etc.
- **Numerosity Reduction** - In this method, various techniques can be applied to reduce the volume of data by choosing alternative smaller representations of the data. For example, a variable can be approximated by a regression model, and instead of storing the entire variable, we can store the regression model to approximate it.
- **Data Compression** - In this method, data is compressed. Data Compression can be lossless or lossy depending on whether the information is lost or not during compression.

**Data Transformation**

Data Transformation is a process of converting data into a format that helps in building efficient ML models and deriving better insights. A few of the most common methods for Data Transformation include -

- **Smoothing** - Data Smoothing is used to remove noise in the dataset, and it helps identify important features and detect patterns. Therefore, it can help in predicting trends or future events.

- **Aggregation** - Data Aggregation is the process of transforming large volumes of data into an organized and summarized format that is more understandable and comprehensive. For example, a company may look at monthly sales data of a product instead of raw sales data to understand its performance better and forecast future sales.
- **Discretization** - Data Discretization is a process of converting numerical or continuous variables into a set of intervals/bins. This makes data easier to analyze. For example, the age features can be converted into various intervals such as (0-10, 11-20, ..) or (child, young, …).
- **Normalization** - Data Normalization is a process of converting a numeric variable into a specified range such as [-1,1], [0,1], etc. A few of the most common approaches to performing normalization are Min-Max Normalization, Data Standardization or Data Scaling, etc.

**Applications of Data Pre-processing**

Data Pre-processing is important in the early stages of a Machine Learning and AI application development lifecycle. A few of the most common usage or application include -

- **Improved Accuracy of ML Models** - Various techniques used to preprocess data, such as Data Cleaning, Transformation ensure that data is complete, accurate, and understandable, resulting in efficient and accurate ML models.
- **Reduced Costs** - Data Reduction techniques can help companies save storage and compute costs by reducing the volume of the data
- **Visualization** - Preprocessed data is easily consumable and understandable that can be further used to build dashboards to gain valuable insights.

**2.2. Reading Data:**

Reading data is a fundamental step in the data science pipeline, setting the stage for analysis, modeling, and interpretation. This process involves acquiring, understanding, and preparing data from various sources. A comprehensive understanding of this topic covers several key areas, including the sources of data, methods of data ingestion, challenges faced during data acquisition, and best practices to ensure data quality and reliability.

**Introduction to Reading Data**

Reading data refers to the process of importing and loading data from various sources into a data science environment. This step is crucial as it forms the foundation upon which all subsequent analysis is built. The ability to efficiently and accurately read data is essential for successful data science projects.

**Sources of Data**

1. **Structured Data**
    - **Databases**: Relational databases (e.g., MySQL, PostgreSQL) and NoSQL databases (e.g., MongoDB).
    - **Spreadsheets**: CSV, Excel files.
2. **Unstructured Data**
    - **Text Files**: Plain text, JSON, XML.

3. **Multimedia:** Images, audio, video files.
4. **Semi-Structured Data**
5. **Logs:** Server logs, application logs.
6. **Web Data:** HTML pages, web scraping outputs.
7. **APIs and Streaming Data**
8. **APIs: RESTful APIs, SOAP APIs for accessing web services.**
9. **Streaming:** Real-time data from sources like Kafka, MQTT.

**Methods of Data Ingestion**

1. **Direct File Reading**
   - **Python Libraries**: pandas (read_csv, read_excel), openpyxl for Excel, json for JSON files.
   - **R Functions**: read.csv, readxl for Excel files.
2. **Database Connectivity**
   - **SQL Queries**: Using sqlite3, SQLAlchemy in Python or DBI in R to connect and query databases.
   - **ORM Tools**: Object-Relational Mapping tools like Django ORM, SQLAlchemy for Python.
3. **Web Scraping**
   - **Libraries**: BeautifulSoup, Scrapy, Selenium for Python.
   - **Tools**: rvest package in R for extracting data from HTML.
4. **API Integration**
   - **Libraries**: requests, http.client in Python, httr in R.
   - **Authentication**: Handling API keys, OAuth tokens for secure data access.
5. **Streaming Data**
   - **Frameworks**: Apache Kafka, Apache Spark Streaming.
   - **Libraries**: kafka-python, pyspark for handling real-time data streams.

**Challenges in Reading Data**

1. **Data Quality**
   - **Missing Data**: Handling null values, imputation techniques.
   - **Inconsistent Data**: Standardizing formats, correcting errors.
2. **Data Volume**
   - **Scalability**: Efficiently reading large datasets using chunking, parallel processing.
3. **Data Variety**
   - **Integration**: Combining data from diverse sources with different formats.
4. **Data Velocity**
   - **Real-Time Processing**: Ensuring timely ingestion and processing of streaming data.
5. **Data Security and Privacy**
   - **Compliance**: Adhering to regulations like GDPR, HIPAA during data acquisition.
   - **Encryption**: Securing data in transit and at rest.

**Best Practices for Reading Data**

1. **Data Profiling**

    - **Initial Assessment**: Understanding the structure, summary statistics, and quality of data before deep analysis.
2. **Efficient Data Handling**
    - **Memory Management**: Using data types efficiently, processing in chunks for large datasets.
    - **Optimized Libraries**: Leveraging high-performance libraries like dask for out-of-core computations.
3. **Documentation and Metadata**
    - **Data Dictionaries**: Maintaining clear documentation of data sources, formats, and fields.
    - **Version Control**: Tracking changes to data ingestion scripts and datasets.
4. **Automated Ingestion Pipelines**
    - **ETL Tools**: Using Extract, Transform, and Load tools like Apache NiFi, Talend.
    - **Scheduling**: Automating data ingestion with tools like Apache Airflow, Cron jobs.
5. **Error Handling and Logging**
    - **Robustness**: Implementing try-catch blocks, logging errors for debugging.

**2.3. Selecting Data:**

Selecting data in data science involves the process of identifying and extracting relevant subsets of data from larger datasets for analysis, modeling, or visualization. This step is crucial for focusing on specific questions or objectives and ensuring that the analysis is based on the most pertinent information. Below is an in-depth discussion of selecting data in data science, covering key concepts, methods, challenges, and best practices.

**Key Concepts in Selecting Data**

- **Relevance**: Choosing data that is directly related to the problem or question at hand.
- **Validity**: Ensuring that the selected data is accurate, reliable, and representative of the underlying phenomena.
- **Completeness**: Including all necessary variables or attributes required for the analysis.
- **Quality**: Selecting data that meets certain quality standards, free from errors, inconsistencies, or biases.
- **Dimensionality**: Considering the number of features or dimensions in the dataset and selecting only those that contribute meaningfully to the analysis.
- **Balance**: Ensuring that the selected data is representative across different classes or categories, especially in classification tasks.
- **Size**: Balancing the trade-off between data size and computational resources available for analysis.

**Methods for Selecting Data**

- **Querying Databases**: Using SQL queries to filter and retrieve specific rows or columns from relational databases.
- **Subsetting**: Extracting subsets of data based on specific criteria or conditions, such as time periods, geographical regions, or categorical variables.

- **Feature Selection**: Identifying and selecting the most relevant features or variables based on statistical techniques, domain knowledge, or feature importance scores.
- **Dimensionality Reduction**: Applying techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dimensionality of the dataset while preserving important information.
- **Sampling**: Taking random samples or stratified samples from the dataset to reduce computational complexity or to balance class distributions.
- **Manual Review**: Inspecting the data visually or through exploratory data analysis (EDA) to identify patterns, outliers, or anomalies and selecting data accordingly.

**Challenges in Selecting Data**

- **Data Quality Issues:** Dealing with missing values, outliers, errors, or inconsistencies that may affect the selection process.
- **Imbalanced Datasets:** Ensuring that the selected data adequately represents all classes or categories, especially in classification tasks with imbalanced class distributions.
- **Biases:** Addressing biases in the data that may result from sampling methods, data collection processes, or underlying societal factors.
- **Scalability:** Handling large datasets efficiently without sacrificing the quality or representativeness of the selected data.
- **Complexity:** Dealing with high-dimensional data or datasets with complex structures that require advanced techniques for selection.
- **Resource Constraints:** Working within the limitations of available computational resources, especially when dealing with large-scale data processing or analysis tasks.

**Best Practices for Selecting Data**

- **Define Clear Objectives:** Clearly define the research questions, objectives, or hypotheses that guide the selection process.
- **Understand the Data:** Gain a deep understanding of the dataset, its characteristics, and its limitations before selecting data subsets.
- **Iterative Approach:** Take an iterative approach to data selection, refining criteria or methods based on initial exploratory analysis or feedback.
- **Document the Process:** Document the data selection process, including criteria, methods, and rationale, to ensure transparency and reproducibility.
- **Validate Selection:** Validate the selected data subsets through cross-validation, hold-out validation, or comparison with external sources to ensure their reliability and generalizability.
- **Address Bias:** Be aware of potential biases in the data and take steps to mitigate them through careful sampling, preprocessing, or modeling techniques.

**2.4. Filtering Data:**

Filtering data in data science involves the process of selecting specific subsets of data based on certain conditions or criteria. This step is essential for focusing the analysis on relevant information, removing noise, and preparing the data for further exploration, modeling, or visualization. Here's an overview of how filtering data is typically done in data science:

**1. Filter by Conditions:**

**a. Numerical Filtering:**

- **Range Selection**: Selecting data points within a specified numerical range.
- **Thresholds**: Filtering data based on numerical thresholds (e.g., values greater than a certain threshold).
- **Comparison Operators**: Using comparison operators (>, <, ==) to filter data.

**b. Text Filtering:**

- **Substring Matching:** Filtering data based on whether a text column contains a specific substring.
- **Regular Expressions:** Using regular expressions to perform complex text pattern matching.

**c. Categorical Filtering:**

- **Membership**: Selecting data points belonging to specific categories or groups.
- **One-Hot Encoding**: Filtering based on binary encoded categorical variables.

**2. Filter by Rows or Columns:**

**a. Row Filtering:**

- **Boolean Indexing:** Using boolean masks to filter rows based on conditions.
- **Querying:** Using query functions to filter rows based on conditions.

**b. Column Filtering:**

- **Selecting Columns**: Choosing specific columns of interest for analysis.
- **Dropping Columns**: Removing columns that are not relevant to the analysis.

**3. Filter by Data Quality:**

**a. Missing Values:**

- **Removal:** Removing rows or columns with missing values.
- **Imputation:** Filling missing values with a suitable estimate (e.g., mean, median, mode).

**b. Outliers:**

- **Statistical Methods**: Identifying outliers using statistical methods such as z-score or IQR (Interquartile Range).
- **Visualization**: Visualizing data distributions to detect outliers visually.

**4. Filter by Time:**

**a. Date Filtering:**

- **Date Range Selection:** Filtering data based on a specified time period or date range.
- **Time Windows:** Filtering data within specific time windows or intervals.

**5. Filter by Data Type:**

**a. Data Type Filtering:**

- **Numeric Data**: Filtering based on numeric data types (integers, floats).
- **Categorical Data**: Filtering based on categorical data types (strings, categories).

**6. Advanced Filtering Techniques:**

**a. Conditional Filtering:**

- **Multiple Conditions:** Filtering data based on multiple conditions combined with logical operators (AND, OR).
- **Nested Conditions:** Applying nested conditions for complex filtering requirements.

**b. Custom Functions:**

- **User-defined Functions**: Defining custom filtering functions to encapsulate complex logic.
- **Lambda Functions**: Using lambda functions for simple filtering operations.

**7. Filter in Different Environments:**

**a. Programming Languages:**

- **Python:** Using libraries such as pandas, NumPy, and scikit-learn for filtering data.
- **R:** Utilizing packages like dplyr and tidyr for data manipulation and filtering.

**b. SQL:**

- **Query Language**: Writing SQL queries to filter data directly from databases (e.g., SELECT WHERE clauses).

**2.5. Filtering Missing Values:**

Filtering missing values, also known as handling missing data, is a crucial step in data preprocessing in data science. Missing values can adversely affect the quality and reliability of analysis results if not handled properly. Here's how missing values are typically filtered in data science:

**1. Identification of Missing Values:**

**a. Null Values:**

- **Identifying Nulls**: Detecting missing values represented as NaN (Not a Number) or NULL in the dataset.
- **Pandas**: Using isnull() or isna() functions in Python's pandas library to identify missing values.

**b. Placeholder Values:**

- **Custom Codes:** Identifying missing values represented by custom codes or placeholders in the dataset.

**2. Visualization and Summary Statistics:**

**a. Heatmaps:**

- **Visualizing Missingness**: Using heatmaps to visualize the presence of missing values across variables.
- **Seaborn**: Utilizing seaborn's heatmap() function in Python for visualizing missing data patterns.

**b. Summary Statistics:**

- **Missingness Summary:** Calculating summary statistics such as the percentage of missing values per variable.
- **Pandas:** Employing functions like info(), describe(), or isnull().sum() in pandas for generating summary statistics.

**3. Filtering Strategies:**

**a. Removal of Missing Values:**

- **Dropping Rows**: Removing rows containing missing values using dropna() function in pandas.
- **Dropping Columns**: Eliminating columns with a high proportion of missing values using drop() function.

**b. Imputation Techniques:**

- **Mean/Median Imputation:** Replacing missing values with the mean or median of the respective column.
- **Mode Imputation:** Imputing missing categorical values with the mode (most frequent) value.
- **Pandas: Utilizing** fillna() function in pandas to perform imputation.

**c. Advanced Imputation Methods:**

- **K-Nearest Neighbors (KNN)**: Imputing missing values based on the values of the nearest neighbors in feature space.
- **Multiple Imputation**: Generating multiple imputed datasets to account for uncertainty in imputed values.
- **Scikit-Learn**: Using SimpleImputer from scikit-learn library for imputation.

**d. Domain-Specific Imputation:**

- **Contextual Imputation:** Imputing missing values based on domain-specific knowledge or business rules.
- **Manual Inspection:** Inspecting missing values manually to determine appropriate imputation strategies.

**4. Handling Large Datasets:**

**a. Sampling:**

- **Random Sampling**: Sampling a subset of the data for manual inspection and imputation.
- **Scalable Methods**: Employing scalable imputation methods for large datasets (e.g., iterative imputation).

**b. Parallel Processing:**

- **Distributed Computing:** Utilizing distributed computing frameworks (e.g., Apache Spark) for parallel imputation.

**5. Evaluation and Validation:**

**a. Impact Assessment:**

- **Evaluation Metrics**: Assessing the impact of missing value handling techniques on analysis results using appropriate evaluation metrics.
- **Cross-Validation**: Performing cross-validation to evaluate the robustness of imputation methods.

**b. Sensitivity Analysis:**

- **Sensitivity Testing:** Conducting sensitivity analysis to evaluate the sensitivity of analysis results to different missing value handling strategies.

**2.6. Manipulating data:**

Manipulating data is a fundamental part of the data science workflow. It involves transforming raw data into a format that is suitable for analysis. This process includes cleaning, reshaping, aggregating, and merging data, among other operations. Effective data manipulation is essential for accurate and efficient analysis. Here's an in-depth look at the key concepts, methods, and best practices for manipulating data in data science:

**Key Concepts in Data Manipulation**

1. **Data Cleaning**: Removing or correcting errors and inconsistencies in the data.
2. **Data Transformation**: Changing the format or structure of the data.
3. **Data Aggregation**: Summarizing data by grouping and calculating aggregate statistics.
4. **Data Merging**: Combining data from different sources.
5. **Feature Engineering**: Creating new features from existing data to improve model performance.

**Methods of Data Manipulation**

**1. Data Cleaning**

- **Handling Missing Values**:
  - **Remove:** Drop rows or columns with missing values.
  - **Impute:** Fill missing values with mean, median, mode, or other values.
  - **Interpolate:** Use interpolation methods for filling missing values in time series data.
- **Removing Duplicates**:
  - **Drop Duplicates:** Remove duplicate rows using functions like drop_duplicates() in pandas.
- **Correcting Errors**:
  - **Data Validation:** Use domain knowledge to correct errors in the data.
  - **Regex:** Use regular expressions to clean text data.

**2. Data Transformation**

- **Normalization/Standardization**:
  - **Scaling:** Scale data to a standard range (e.g., 0 to 1) or to have zero mean and unit variance.
- **Encoding Categorical Variables**:
  - **Label Encoding:** Convert categorical labels to numerical values.
  - **One-Hot Encoding:** Create binary columns for each category.
- **Date and Time Manipulation**:
  - **Extracting Components:** Extract year, month, day, hour, etc., from datetime objects.
  - **Datetime Conversion:** Convert strings to datetime objects.

**3. Data Aggregation**

- **Grouping Data**:

    - **Group By:** Group data by one or more columns and compute aggregate statistics (e.g., sum, mean).
- **Pivot Tables**:
    - **Pivot:** Reshape data to summarize information using pivot tables.

**4. Data Merging**

- **Joining Data:**
    - **Merge/Join:** Combine data from different sources based on common keys using functions like merge() in pandas.
    - **Concatenate:** Stack data vertically or horizontally.

**5. Feature Engineering**

- **Creating New Features**:
    - **Mathematical Transformations**: Apply mathematical functions (e.g., log, square root) to create new features.
    - **Interaction Terms**: Create features representing interactions between variables.
- **Binning**:
    - **Discretization:** Convert continuous variables into categorical bins.

**Tools for Data Manipulation**

**1. Python**

- **pandas:** The go-to library for data manipulation in Python. Provides functions for data cleaning, transformation, aggregation, and merging.
- **NumPy**: Useful for numerical operations and array manipulations.
- **SciPy:** Provides additional functionality for mathematical operations and statistics.

**2. R**

- **dplyr:** A powerful package for data manipulation in R. Provides functions for filtering, selecting, and summarizing data.
- **tidyr:** Complementary to dplyr, used for reshaping and tidying data.
- **data.table:** High-performance data manipulation package for large datasets.

**Best Practices for Data Manipulation**

1. **Understand the Data**:
    - Perform exploratory data analysis (EDA) to understand the structure, distributions, and relationships within the data.
2. **Write Clean and Reproducible Code**:
    - Document each step of the data manipulation process.
    - Use version control systems like Git to track changes.
3. **Optimize Performance**:
    - For large datasets, use efficient data structures and algorithms.
    - Leverage parallel processing and vectorized operations where possible.
4. **Validate Changes**:

- After each manipulation step, check the data to ensure that the operations have been performed correctly.
- Use summary statistics and visualizations to validate changes.

5. **Maintain Data Integrity**:
   - Ensure that data manipulation steps do not introduce errors or inconsistencies.

**2.7. Sorting:**

- Sorting data is a fundamental operation in data science that involves arranging data in a particular order, typically ascending or descending, based on one or more attributes. Sorting is crucial for data analysis, visualization, and preparation for further operations. Here's a detailed look at sorting data in data science:

**Key Concepts in Sorting Data**

- **Ordering**: Arranging data in ascending or descending order based on the values of one or more columns.
- **Stability**: Maintaining the relative order of records with equal keys (values) after sorting.
- **Multi-Level Sorting**: Sorting data based on multiple columns, where the order of precedence is specified.

**Methods for Sorting Data**

- Single-Column Sorting**: Sorting data based on the values of a single column.**
- Multi-Column Sorting**: Sorting data based on the values of multiple columns, with a defined order of precedence.**

**Sorting in Different Environments**

**Python (using pandas)**

- Pandas is a powerful Python library for data manipulation and analysis. Here's how to sort data using pandas:

**Single-Column Sorting**

```
import pandas as pd

# Sample DataFrame

data = {

'id': [1, 2, 3, 4, 5],

'name': ['Alice', 'Bob', 'Charlie', 'David', 'Edward'],
```

```
'age': [25, 35, 45, 20, 30]
}
 df = pd.DataFrame(data)
# Sort by age in ascending order
 sorted_df = df.sort_values(by='age')
print(sorted_df)
```

**R (using dplyr)**

dplyr is a powerful R package for data manipulation. Here's how to sort data using dplyr:

**Single-Column Sorting**

```
library(dplyr)
# Sample data frame
 df <- data.frame(
id = c(1, 2, 3, 4, 5),
name = c('Alice', 'Bob', 'Charlie', 'David', 'Edward'),
age = c(25, 35, 45, 20, 30)
 )
 # Sort by age in ascending order
sorted_df <- df %>% arrange(age)
 print(sorted_df)
```

**SQL**

SQL (Structured Query Language) is used to interact with relational databases. Here's how to sort data using SQL:

**Single-Column Sorting**

```
SELECT id, name, age
```

```
FROM table_name

ORDER BY  age ASC;
```

**Challenges in Sorting Data**

1. **Handling Missing Values**: Deciding whether to place missing values at the beginning or end of the sorted list.
2. **Performance**: Sorting large datasets efficiently without consuming excessive memory or time.
3. **Locale-Specific Sorting**: Sorting text data based on locale-specific rules (e.g., case sensitivity, special characters).
4. **Stability**: Ensuring that the sort is stable, meaning that the relative order of records with equal keys remains unchanged.

**Best Practices for Sorting Data**

- **Understand Data Types:** Ensure you understand the data types of the columns being sorted, as numerical, textual, and date data types are handled differently.
- **Handle Missing Values**: Explicitly define how to handle missing values during sorting to avoid unexpected results.
- **Optimize Performance:** For large datasets, consider using optimized sorting algorithms or tools that can handle large-scale data efficiently.
- **Multi-Level Sorting:** When performing multi-level sorting, clearly define the order of precedence for columns.
- **Maintain Stability:** If the order of records with equal keys is important, use a stable sorting algorithm.

**2.8. Grouping:**

Grouping data is a powerful technique in data science used to aggregate and summarize data, making it easier to analyze and derive insights from large datasets. Grouping involves splitting the data into subsets based on one or more criteria and then applying aggregate functions to each subset. This method is particularly useful for identifying patterns, trends, and relationships within the data.

**Key Concepts in Grouping Data**

- **Grouping**: Splitting the dataset into subsets based on the values of one or more columns.
- **Aggregation**: Applying functions to each group to compute summary statistics such as mean, sum, count, max, and min.
- **Transformation**: Applying functions to each group to transform the data.
- **Filtering**: Selecting groups based on aggregate criteria.

**Grouping in Different Environments**

**Python (using pandas)**

- **Pandas is a powerful library for data manipulation in Python. Here's how to perform grouping using pandas:**

**Grouping by a Single Column**

```
import pandas as pd
# Sample DataFrame
 data = { 'id': [1, 2, 3, 4, 5],
 'name': ['Alice', 'Bob', 'Charlie', 'David', 'Edward'],
 'age': [25, 35, 45, 20, 30],
'sales': [100, 200, 300, 400, 500]
 }
 df = pd.DataFrame(data)
# Group by 'name' and calculate the sum of 'sales'
 grouped_df = df.groupby('name')['sales'].sum().reset_index()
print(grouped_df)
```

**R (using dplyr)**

dplyr is a powerful R package for data manipulation. Here's how to perform grouping using dplyr:

**Grouping by a Single Column**

```
library(dplyr)
# Sample data frame
df <- data.frame( id = c(1, 2, 3, 4, 5),
 name = c('Alice', 'Bob', 'Charlie', 'David', 'Edward'),
age = c(25, 35, 45, 20, 30),
sales = c(100, 200, 300, 400, 500) )
# Group by 'name' and calculate the sum of 'sales'
grouped_df <- df %>% group_by(name) %>% summarise(sales_sum = sum(sales))
 print(grouped_df)
```

**SQL**

SQL (Structured Query Language) is used to interact with relational databases. Here's how to perform grouping using SQL:

**Grouping by a Single Column**

```
SELECT name, SUM(sales) AS sales_sum
 FROM table_name
GROUP BY name;
```

**Advanced Grouping Techniques**

1. **Group Filtering (Having Clause in SQL)**:
    - Filtering groups based on aggregate criteria.
    - Example in SQL:

```
sql
SELECT name, SUM(sales) AS sales_sum
FROM table_name
GROUP BY name
 HAVING SUM(sales) > 300;
```

**Transformations within Groups**:

- Applying transformations or calculations within each group.
- Example in pandas:

```
df['sales_cumsum'] = df.groupby('name')['sales'].cumsum()
```

**Hierarchical Grouping**:

- Grouping data at multiple hierarchical levels.
- Example in pandas:

```
grouped_df = df.groupby(['name', 'age'])['sales'].sum().unstack()
```

**Window Functions (SQL)**:

- Applying window functions for cumulative sums, moving averages, etc., within groups.
- Example in SQL:

```
SELECT name, sales, SUM(sales) OVER (PARTITION BY name ORDER BY id) AS
cumulative_sales FROM table_name;
```

**Challenges and Best Practices**

1. **Handling Large Datasets**:
    - Ensure efficient memory usage and performance.
    - Consider using optimized libraries or distributed computing frameworks.
2. **Dealing with Missing Values**:
    - Handle missing values appropriately before grouping to avoid inaccurate results.
3. **Choosing the Right Aggregation Functions**:
    - Select aggregation functions that align with the analysis objectives.
4. **Interpreting Results**:
    - Ensure the results of grouping and aggregation are interpretable and meaningful for the analysis context.

**2.9. Rearranging**

Rearranging data in data science involves organizing, transforming, and restructuring raw data into a format that is more suitable for analysis or modeling. This process, often referred to as data manipulation or data wrangling, is a fundamental step in the data science workflow. Here are some common techniques and tools used for rearranging data:

1. **Data Cleaning**: This involves identifying and handling missing values, removing duplicates, correcting inconsistencies, and dealing with outliers to ensure that the data is accurate and reliable for analysis.
2. **Reshaping Data**: Reshaping data involves transforming its structure, typically by converting it from wide to long format or vice versa. This can be done using functions like `pivot`, `melt`, `stack`, and `unstack` in Python libraries like Pandas.
3. **Subset Selection**: Subset selection involves extracting specific rows or columns from a dataset based on certain criteria or conditions. This can be achieved using filtering or selection functions like `loc` and `iloc` in Pandas.
4. **Sorting**: Sorting data involves arranging it in a particular order based on one or more variables, such as alphabetical order, numerical order, or chronological order. Sorting can be done using the `sort_values` function in Pandas.
5. **Joining and Merging**: Joining and merging involve combining multiple datasets based on common keys or columns. This is useful for integrating data from different sources or combining related datasets. Functions like `merge` and `concatenate` in Pandas are commonly used for this purpose.
6. **Grouping and Aggregating**: Grouping and aggregating data involve splitting it into groups based on certain variables and then applying summary statistics or aggregation functions to each group. This is commonly used for analyzing data at different levels of granularity. Functions like `groupby` and `agg` in Pandas are used for grouping and aggregating data.
7. **Feature Engineering**: Feature engineering involves creating new features or variables from existing data that may be more informative for modeling or analysis. This can include transformations, scaling, encoding categorical variables, or creating interaction terms.
8. **Time Series Manipulation**: For time series data, rearranging may involve resampling (changing the frequency of the data), shifting (moving data forward or backward in time), or calculating differences or percentage changes.

These are just a few examples of the many techniques and tools available for rearranging data in data science. The choice of technique depends on the specific requirements of the analysis and the characteristics of the dataset. Python libraries like Pandas, NumPy, and scikit-learn provide extensive support for data manipulation, while tools like SQL and Excel are also commonly used, depending on the context and requirements.

### 2.10. Ranking:

Ranking data in data science involves assigning a numerical value to each item in a dataset based on a specific criterion or set of criteria. Ranking is commonly used in various analytical tasks such as identifying top performers, prioritizing items, or evaluating the relative importance of different elements in a dataset. Here are some common techniques for ranking data:

1. **Simple Ranking**: Assigning ranks to items based on a single criterion, such as their numerical value or performance score. This can be done using built-in functions like `rank` in Python's Pandas library.
2. **Percentile Ranking**: Dividing the data into equal-sized groups (percentiles) and assigning ranks based on the percentile each item falls into. This is useful for comparing the relative position of each item within the distribution of the data.
3. **Rank Aggregation**: Combining rankings from multiple sources or criteria to generate a consensus ranking. This can involve methods like averaging ranks, Borda count, or Markov chain models.
4. **Tie-breaking**: Handling ties that occur when multiple items have the same rank. Tie-breaking strategies include methods like randomization, averaging ranks, or using secondary criteria.
5. **Ranking with Weighted Criteria**: Assigning ranks based on multiple criteria with different weights or importance levels. This involves aggregating scores from different criteria using weighted averages or other aggregation methods.
6. **Ranking Time Series Data**: Ranking data over time to identify trends, fluctuations, or changes in rank positions. This can involve techniques like ranking by moving average or ranking by trend strength.
7. **Ranking in Machine Learning**: In machine learning, ranking is often used in tasks such as recommender systems, where items are ranked based on their relevance to a user. Learning-to-rank algorithms, such as RankNet or LambdaMART, are commonly used for these tasks.
8. **Dynamic Ranking**: Updating ranks in real-time as new data becomes available. This is common in applications such as online advertising or financial trading, where rankings need to be continuously updated based on changing conditions.

## UNIT – III

## ESSENTIALS OF R

### 3.1. R Basics:

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac. This programming language was named **R**, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of the Bell Labs Language **S**.

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

R is free software distributed under a GNU-style copy left, and an official part of the GNU project called **GNU S**.

**Evolution of R**

R was initially written by **Ross Ihaka** and **Robert Gentleman** at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

- A large group of individuals has contributed to R by sending code and bug reports.
- Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

**Features of R**

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R –

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

**R Data Types**

Different forms of data that can be saved and manipulated are defined and categorized using data types in computer languages, including R. Each R data type has unique properties and associated operations.

**What are R Data types?**

**R Data types** are used to specify the kind of data that can be stored in a variable.

For effective memory consumption and precise computation, the right data type must be selected.

Each R data type has its own set of regulations and restrictions.

Variables are not needed to be declare with a data type in R, data type even can be changed.

**Example of R data Type:**

#numeric data type

var <- 30

#integer data type

var <- 80L

**Data Types in R Programming Language**

Each variable in R has an associated data type. Each R-Data Type requires different amounts of memory and has some specific operations which can be performed over it.

**Data Types in R are:**

numeric – (3,6.7,121)

Integer – (2L, 42L; where 'L' declares this as an integer)

logical – ('True')

complex – (7 + 5i; where 'i' is imaginary number)

character – (“a”, “B”, “c is third”, “69”)

raw – ( as.raw(55); raw creates a raw vector of the specified length)

R Programming language has the following basic R-data types and the following table shows the data type and the values that each data type can take.

| Basic Data Types | Values | Examples |
|---|---|---|
| Numeric | Set of all real numbers | "numeric_value <- 3.14" |
| Integer | Set of all integers, Z | "integer_value <- 42L" |
| Logical | TRUE and FALSE | "logical_value <- TRUE" |
| Complex | Set of complex numbers | "complex_value <- 1 + 2i" |
| Character | “a”, “b”, “c”, …, “@”, “#”, “$”, …., “1”, “2”, …etc | "character_value <- "Hello Geeks" |
| raw | as.raw() | "single_raw <- as.raw(255)" |

**1. Numeric Data type in R**

Decimal values are called numeric in R. It is the default R data type for numbers in R.

If you assign a decimal value to a variable x as follows, x will be of numeric type.

Real numbers with a decimal point are represented using this data type in R. It uses a format for double-precision floating-point numbers to represent numerical values.

```
# A simple R program

# to illustrate Numeric data type

 # Assign a decimal value to x

x = 5.6

 # print the class name of variable

print(class(x))

# print the type of variable

print(typeof(x))
```

**Output**

[1] "numeric"

[1] "double"

Even if an integer is assigned to a variable y, it is still saved as a numeric value.

```
# A simple R program

# to illustrate Numeric data type

 # Assign an integer value to y

y = 5

 # print the class name of variable

print(class(y))

 # print the type of variable

print(typeof(y))
```

**Output**

[1] "numeric"

[1] "double"

When R stores a number in a variable, it converts the number into a “double” value or a decimal type with at least two decimal places.

This means that a value such as “5” here, is stored as 5.00 with a type of double and a class of numeric. And also y is not an integer here can be confirmed with the **is.integer()** function.

```
# A simple R program

# to illustrate Numeric data type

# Assign a integer value to y

y = 5

 # is y an integer?

print(is.integer(y))
```

**Output**

[1] FALSE

## 2. Integer Data type in R

R supports integer data types which are the set of all integers.

You can create as well as convert a value into an integer type using the **as.integer()** function.

You can also use the capital 'L' notation as a suffix to denote that a particular value is of the integer R data type.

```
# A simple R program

# to illustrate integer data type

 # Create an integer value

x = as.integer(5)

 # print the class name of x

print(class(x))

 # print the type of x

print(typeof(x))

 # Declare an integer by appending an L suffix.

y = 5L

 # print the class name of y

print(class(y))

# print the type of y

print(typeof(y))
```

**Output**

```
[1] "integer"

[1] "integer"

[1] "integer"

[1] "integer"
```

## 3. Logical Data type in R

R has logical data types that take either a value of **true** or **false**.

A logical value is often created via a comparison between variables.

Boolean values, which have two possible values, are represented by this R data type: FALSE or TRUE

```
# A simple R program

# to illustrate logical data type

 # Sample values

x = 4

y = 3


# Comparing two values

z = x > y


# print the logical value

print(z)


# print the class name of z

print(class(z))


# print the type of z

print(typeof(z))
```

**Output**

```
[1] TRUE

[1] "logical"

[1] "logical"
```

### 4. Complex Data type in R

R supports complex data types that are set of all the complex numbers. The complex data type is to store numbers with an imaginary component.

```
# A simple R program

# to illustrate complex data type


# Assign a complex value to x

x = 4 + 3i


# print the class name of x

print(class(x))


# print the type of x

print(typeof(x))
```

**Output**

```
[1] "complex"

[1] "complex"
```

### 5. Character Data type in R

R supports character data types where you have all the alphabets and special characters.

It stores character values or strings. Strings in R can contain alphabets, numbers, and symbols.

The easiest way to denote that a value is of character type in R data type is to wrap the value inside single or double inverted commas.

```
# A simple R program

# to illustrate character data type


# Assign a character value to char

char = "Geeksforgeeks"


# print the class name of char
```

```
print(class(char))


# print the type of char

print(typeof(char))
```

**Output**

```
[1] "character"

[1] "character"
```

There are several tasks that can be done using R data types. Let's understand each task with its action and the syntax for doing the task along with an R code to illustrate the task.

### 6. Raw data type in R

To save and work with data at the byte level in R, use the raw data type. By displaying a series of unprocessed bytes, it enables low-level operations on binary data. Here are some speculative data on R's raw data types:

```
# Create a raw vector

x <- as.raw(c(0x1, 0x2, 0x3, 0x4, 0x5))

print(x)
```

**Output**

```
[1] 01 02 03 04 05
```

Five elements make up this raw vector x, each of which represents a raw byte value.

### Find Data Type of an Object in R

To find the data type of an object you have to use **class()** function. The syntax for doing that is you need to pass the object as an argument to the function **class()** to find the data type of an object.

**Syntax**

```
class(object)
```

**Example**

```
# A simple R program
```

```
# to find data type of an object

 # Logical

print(class(TRUE))

 # Integer

print(class(3L))

 # Numeric

print(class(10.5))


# Complex

print(class(1+2i))

 # Character

print(class("12-04-2020"))
```

**Output**

```
[1] "logical"

[1] "integer"

[1] "numeric"

[1] "complex"

[1] "character"
```

**Type verification**

You can verify the data type of an object, if you doubt about it's data type.

To do that, you need to use the prefix "is." before the data type as a command.

**Syntax:**

```
is.data_type(object)
```

**Example**

```
# A simple R program

# Verify if an object is of a certain datatype


# Logical

print(is.logical(TRUE))


# Integer

print(is.integer(3L))


# Numeric

print(is.numeric(10.5))


# Complex

print(is.complex(1+2i))


# Character

print(is.character("12-04-2020"))


print(is.integer("a"))


print(is.numeric(2+3i))
```

**Output**

```
[1] TRUE

[1] TRUE

[1] TRUE

[1] TRUE
```

[1] TRUE

[1] FALSE

[1] FALSE

**Coerce or Convert the Data Type of an Object to Another**

The process of altering the data type of an object to another type is referred to as coercion or data type conversion. This is a common operation in many programming languages that is used to alter data and perform various computations.

When coercion is required, the language normally performs it automatically, whereas conversion is performed directly by the programmer.

Coercion can manifest itself in a variety of ways, depending on the R programming language and the context in which it is employed.

In some circumstances, the coercion is implicit, which means that the language will change one type to another without the programmer having to expressly request it.

**Syntax**

as.data_type(object)

**Note:** All the coercions are not possible and if attempted will be returning an “NA” value.

**For Detailed Explanation – Data Type Conversion in R**

**Example**

```
# A simple R program

# convert data type of an object to another


# Logical

print(as.numeric(TRUE))


# Integer

print(as.complex(3L))
```

```
# Numeric

print(as.logical(10.5))


# Complex

print(as.character(1+2i))


# Can't possible

print(as.numeric("12-04-2020"))
```

**Output**

```
[1] 1
[1] 3+0i
[1] TRUE
[1] "1+2i"
[1] NA
Warning message:
In print(as.numeric("12-04-2020")) : NAs introduced by coercion
```

**Control Structures**

Control structures in R allow you to control the flow of execution of a series of R expressions. Basically, control structures allow you to put some “logic” into your R code, rather than just always executing the same R code every time. Control structures allow you to respond to inputs or to features of the data and execute different R expressions accordingly.

Commonly used control structures are

- `if` and `else`: testing a condition and acting on it
- `for`: execute a loop a fixed number of times
- `while`: execute a loop *while* a condition is true
- `repeat`: execute an infinite loop (must `break` out of it to stop)
- `break`: break the execution of a loop
- `next`: skip an interation of a loop

Most control structures are not used in interactive sessions, but rather when writing functions or longer expresisons. However, these constructs do not have to be used in functions and it’s a good idea to become familiar with them before we delve into functions.

**if-else:**

The `if-else` combination is probably the most commonly used control structure in R (or perhaps any language). This structure allows you to test a condition and act on it depending on whether it's true or false.

For starters, you can just use the `if` statement.

```
if(<condition>) {
        ## do something
}
## Continue with rest of code
```

The above code does nothing if the condition is false. If you have an action you want to execute when the condition is false, then you need an `else` clause.

```
if(<condition>) {
        ## do something
}
else {
        ## do something else
}
```

You can have a series of tests by following the initial `if` with any number of `else if`s.

```
if(<condition1>) {
        ## do something
} else if(<condition2>)  {
        ## do something different
} else {
        ## do something different
}
```

Here is an example of a valid if/else structure.

```
## Generate a uniform random number
x <- runif(1, 0, 10)
if(x > 3) {
        y <- 10
} else {
        y <- 0
}
```

The value of `y` is set depending on whether `x > 3` or not. This expression can also be written a different, but equivalent, way in R.

```
y <- if(x > 3) {
        10
} else {
        0
}
```

Neither way of writing this expression is more correct than the other. Which one you use will depend on your preference and perhaps those of the team you may be working with.

Of course, the `else` clause is not necessary. You could have a series of if clauses that always get executed if their respective conditions are true.

```
if(<condition1>) {

}

if(<condition2>) {

}
```

**for Loops:**

For loops are pretty much the only looping construct that you will need in R. While you may occasionally find a need for other types of loops, in my experience doing data analysis, I've found very few situations where a for loop wasn't sufficient.

In R, for loops take an interator variable and assign it successive values from a sequence or vector. For loops are most commonly used for iterating over the elements of an object (list, vector, etc.)

```
> for(i in 1:10) {
+         print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

This loop takes the `i` variable and in each iteration of the loop gives it values 1, 2, 3, …, 10, executes the code within the curly braces, and then the loop exits.

The following three loops all have the same behavior.

```
> x <- c("a", "b", "c", "d")
>
> for(i in 1:4) {
+         ## Print out each element of 'x'
+         print(x[i])
+ }
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

The `seq_along()` function is commonly used in conjunction with for loops in order to generate an integer sequence based on the length of an object (in this case, the object `x`).

```
> ## Generate a sequence based on length of 'x'
> for(i in seq_along(x)) {
+         print(x[i])
+ }
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

It is not necessary to use an index-type variable.

```
> for(letter in x) {
+         print(letter)
+ }
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

For one line loops, the curly braces are not strictly necessary.

```
> for(i in 1:4) print(x[i])
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

However, I like to use curly braces even for one-line loops, because that way if you decide to expand the loop to multiple lines, you won't be burned because you forgot to add curly braces (and you *will* be burned by this).

**Nested for loops:**

`for` loops can be nested inside of each other.

```
x <- matrix(1:6, 2, 3)

for(i in seq_len(nrow(x))) {
        for(j in seq_len(ncol(x))) {
                print(x[i, j])
        }
}
```

Nested loops are commonly needed for multidimensional or hierarchical data structures (e.g. matrices, lists). Be careful with nesting though. Nesting beyond 2 to 3 levels often makes it difficult to read/understand the code. If you find yourself in need of a large number of nested loops, you may want to break up the loops by using functions (discussed later).

**while Loops:**

While loops begin by testing a condition. If it is true, then they execute the loop body. Once the loop body is executed, the condition is tested again, and so forth, until the condition is false, after which the loop exits.

```
> count <- 0
> while(count < 10) {
+         print(count)
+         count <- count + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

While loops can potentially result in infinite loops if not written properly. Use with care!

Sometimes there will be more than one condition in the test.

```
> z <- 5
> set.seed(1)
>
> while(z >= 3 && z <= 10) {
+         coin <- rbinom(1, 1, 0.5)
+
+         if(coin == 1) {  ## random walk
+                 z <- z + 1
+         } else {
+                 z <- z - 1
+         }
+ }
> print(z)
[1] 2
```

Conditions are always evaluated from left to right. For example, in the above code, if `z` were less than 3, the second test would not have been evaluated.

**repeat Loops:**

`repeat` initiates an infinite loop right from the start. These are not commonly used in statistical or data analysis applications but they do have their uses. The only way to exit a `repeat` loop is to call `break`.

One possible paradigm might be in an iterative algorith where you may be searching for a solution and you don't want to stop until you're close enough to the solution. In this kind of situation, you often don't know in advance how many iterations it's going to take to get "close enough" to the solution.

```
x0 <- 1
tol <- 1e-8

repeat {
        x1 <- computeEstimate()
```

```
        if(abs(x1 - x0) < tol) {  ## Close enough?
                break
        } else {
                x0 <- x1
        }
}
```

Note that the above code will not run if the `computeEstimate()` function is not defined (I just made it up for the purposes of this demonstration).

The loop above is a bit dangerous because there's no guarantee it will stop. You could get in a situation where the values of `x0` and `x1` oscillate back and forth and never converge. Better to set a hard limit on the number of iterations by using a `for` loop and then report whether convergence was achieved or not.

**next, break:**

`next` is used to skip an iteration of a loop.

```
for(i in 1:100) {
        if(i <= 20) {
                ## Skip the first 20 iterations
                next
        }
        ## Do something here
}
```

`break` is used to exit a loop immediately, regardless of what iteration the loop may be on.

```
for(i in 1:100) {
      print(i)

      if(i > 20) {
              ## Stop loop after 20 iterations
              break
      }
}
```

**Data Frames**

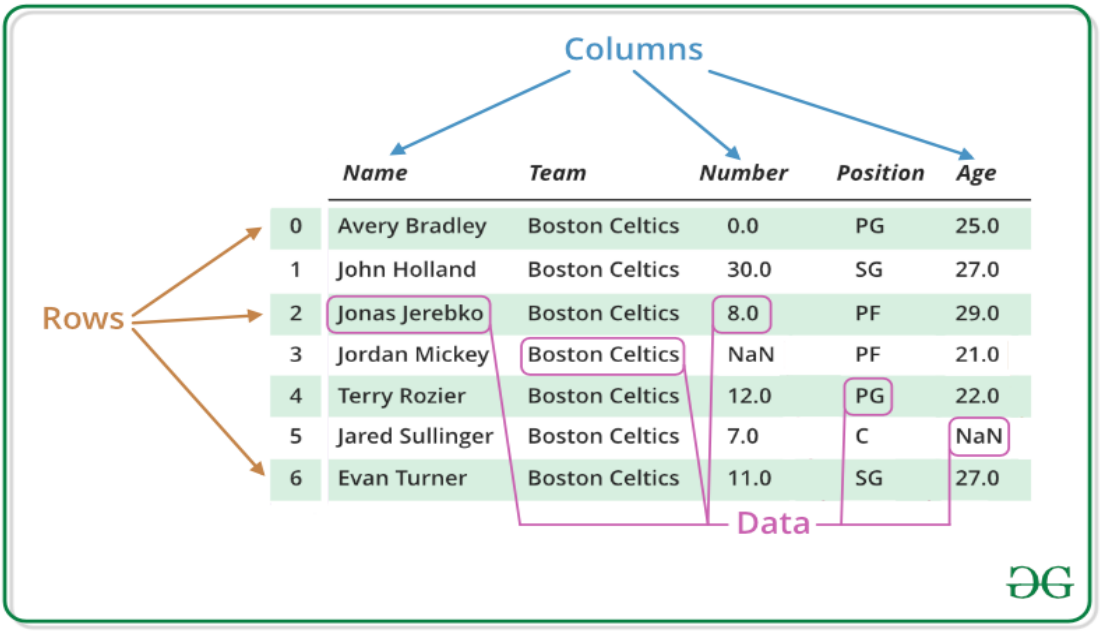
R Programming Language is an open-source programming language that is widely used as a statistical software and data analysis tool. **Data Frames in R Language** are generic data objects of R that are used to store tabular data.

Data frames can also be interpreted as matrices where each column of a matrix can be of different data types. R DataFrame is made up of three principal components, the data, rows, and columns.

**R Data Frames Structure**

As you can see in the image below, this is how a data frame is structured.

The data is presented in tabular form, which makes it easier to operate and understand.




R – Data Frames

**Create Dataframe in R Programming Language**

To create an R data frame use **data.frame()** function and then pass each of the vectors you have created as arguments to the function.

```
# R program to create dataframe

# creating a data frame
friend.data <- data.frame(
    friend_id = c(1:5),
    friend_name = c("Sachin", "Sourav",
                    "Dravid", "Sehwag",
                    "Dhoni"),
    stringsAsFactors = FALSE
)
# print the data frame
print(friend.data)
```

**Output:**

```
  friend_id friend_name
1         1      Sachin
2         2      Sourav
3         3      Dravid
4         4      Sehwag
5         5       Dhoni
```

### Get the Structure of the R Data Frame

One can get the structure of the R data frame using **str()** function in R.

It can display even the internal structure of large lists which are nested. It provides one-liner output for the basic R objects letting the user know about the object and its constituents.

```
# R program to get the
# structure of the data frame

# creating a data frame
friend.data <- data.frame(
    friend_id = c(1:5),
    friend_name = c("Sachin", "Sourav",
                    "Dravid", "Sehwag",
                    "Dhoni"),
    stringsAsFactors = FALSE
)
# using str()
print(str(friend.data))
```

**Output:**

```
'data.frame':    5 obs. of  2 variables:
 $ friend_id  : int  1 2 3 4 5
 $ friend_name: chr  "Sachin" "Sourav" "Dravid" "Sehwag" ...
NULL
```

### Summary of Data in the R data frame

In the R data frame, the statistical summary and nature of the data can be obtained by applying **summary()** function.

It is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

```
# R program to get the
# summary of the data frame

# creating a data frame
friend.data <- data.frame(
    friend_id = c(1:5),
    friend_name = c("Sachin", "Sourav",
                    "Dravid", "Sehwag",
                    "Dhoni"),
    stringsAsFactors = FALSE
)
# using summary()
print(summary(friend.data))
```

**Output:**

```
   friend_id friend_name
 Min.   :1   Length:5
 1st Qu.:2   Class :character
 Median :3   Mode  :character
 Mean   :3
 3rd Qu.:4
 Max.   :5
```

### Extract Data from Data Frame in R

Extracting data from an R data frame means that to access its rows or columns. One can extract a specific column from an R data frame using its column name.

```
# R program to extract
# data from the data frame

# creating a data frame
friend.data <- data.frame(
    friend_id = c(1:5),
    friend_name = c("Sachin", "Sourav",
                    "Dravid", "Sehwag",
                    "Dhoni"),
    stringsAsFactors = FALSE
)

# Extracting friend_name column
result <- data.frame(friend.data$friend_name)
print(result)
```

### Output:

```
  friend.data.friend_name
1                  Sachin
2                  Sourav
3                  Dravid
4                  Sehwag
5                   Dhoni
```

### Access Items in R Data Frame

We can select and access any element from data frame by using single `$` ,brackets `[ ]` or double brackets `[[]]` to access columns from a data frame.

```
# creating a data frame
friend.data <- data.frame(
  friend_id = c(1:5),
  friend_name = c("Sachin", "Sourav",
                  "Dravid", "Sehwag",
                  "Dhoni"),
  stringsAsFactors = FALSE
)

# Access Items using []
friend.data[1]
```

```
# Access Items using [[]]
friend.data[['friend_name']]

# Access Items using $
friend.data$friend_id
```

**Output:**

```
  friend_id
1         1
2         2
3         3
4         4
5         5
Access Items using [[]]
[1] "Sachin" "Sourav" "Dravid" "Sehwag" "Dhoni"
 Access Items using $
[1] 1 2 3 4 5
```

**Feature Engineering in R Programming**

**Feature engineering** is the process of transforming raw data into features that can be used in a machine-learning model. In R programming, feature engineering can be done using a variety of built-in functions and packages.

One common approach to feature engineering is to use the dplyr package to manipulate and summarize data. This package provides functions such as “select()” to select specific columns from a data frame, “filter()” to filter rows based on certain criteria, and “group_by()” to group data by one or more variables and perform aggregate calculations.

Another popular package for feature engineering in R is the tidyr package, which provides functions to reshape and restructure data, such as “gather()” and “spread()” to reshape wide and long format data respectively.

Additionally, you can use functions from base R such as “aggregate()”, “apply()”, “tapply()” to perform calculations on groups of data, or “as.factor()” to convert variables into factors.

Additionally, you can use libraries like caret, which provides various functions to preprocess data, such as normalization, scaling, encoding, and feature selection.

It’s important to note that Feature engineering is an iterative process that requires a good understanding of the data and the problem you are trying to solve, and it’s a crucial step in building a good machine-learning model.

Feature engineering is the most important technique used in creating machine learning models. Feature Engineering is a basic term used to cover many operations that are performed on the variables(features)to fit them into the algorithm. It helps in increasing the accuracy of the model thereby enhancing the results of the predictions. Feature Engineered machine learning models perform better on data than basic machine learning models.

The following aspects of feature engineering are as follows:

- Feature Scaling: **It is done to get the features on the same scale( for eg. Euclidean distance).**
- Feature Transformation: **It is done to normalize the data(feature) by a function.**
- **Feature Construction:** It is done to create new features based on original descriptors to improve the accuracy of the predictive model.
- **Feature Reduction. :** It is done to improve the statistical distribution and accuracy of the predictive model.

**Theory**

The feature Construction method helps in creating new features in the data thereby increasing model accuracy and overall predictions. It is of two types:

- Binning: **Bins are created for continuous variables.**
- Encoding: **Numerical variables or features are formed from categorical variables.**

**Binning**

Binning is done to create bins for continuous variables where they are converted to categorical variables. There are two types of binning: **Unsupervised and Supervised**.

- **Unsupervised Binning** involves Automatic and Manual binning. In Automatic Binning, bins are created without human interference and are created automatically. In Manual Binning, bins are created with human interference and we specify where the bins to be created.
- **Supervised Binning** involves creating bins for the continuous variable while taking the target variable into the consideration also.

**Encoding**

**Encoding is the process in which numerical variables or features are created from categorical variables. It is a widely used method in the industry and in every model building process. It is of two types:** Label Encoding and One-hot Encoding**.**

- **Label Encoding** involves assigning each label a unique integer or value based on alphabetical ordering. It is the most popular and widely used encoding.
- **One-hot Encoding** involves creating additional features or variables on the basis of unique values in categorical variables i.e. every unique value in the category will be added as a new feature.

**Implementation in R**

**The Dataset**

BigMart dataset consists of 1559 products across 10 stores in different cities. Certain attributes of each product and store have been defined. It consists of 12 features i.e

Item_Identifier( is a unique product ID assigned to every distinct item), Item_Weight(includes the weight of the product), Item_Fat_Content(describes whether the product is low fat or not), Item_Visibility(mentions the percentage of the total display area of all products in a store allocated to the particular product), Item_Type(describes the food category to which the item belongs), Item_MRP(Maximum Retail Price (list price) of the product), Outlet_Identifier(unique store ID assigned. It consists of an alphanumeric string of length 6), Outlet_Establishment_Year(mentions the year in which store was established), Outlet_Size(tells the size of the store in terms of ground area covered), Outlet_Location_Type(tells about the size of the city in which the store is located), Outlet_Type(tells whether the outlet is just a grocery store or some sort of supermarket) and Item_Outlet_Sales( sales of the product in the particular store).

```
# Loading data

train = fread("Train_UWu5bXk.csv")

test = fread("Test_u94Q5KV.csv")



# Structure

str(train)
```

**Output:**

```
> str(train)
Classes 'data.table' and 'data.frame':  8523 obs. of  12 variables:
 $ Item_Identifier          : chr  "FDA15" "DRC01" "FDN15" "FDX07" ...
 $ Item_Weight              : num  9.3 5.92 17.5 19.2 8.93 ...
 $ Item_Fat_Content         : chr  "Low Fat" "Regular" "Low Fat" "Regular" ...
 $ Item_Visibility          : num  0.016 0.0193 0.0168 0 0 ...
 $ Item_Type                : chr  "Dairy" "Soft Drinks" "Meat" "Fruits and Vegetables" ...
 $ Item_MRP                 : num  249.8 48.3 141.6 182.1 53.9 ...
 $ Outlet_Identifier        : chr  "OUT049" "OUT018" "OUT049" "OUT010" ...
 $ Outlet_Establishment_Year: int  1999 2009 1999 1998 1987 2009 1987 1985 2002 2007 ...
 $ Outlet_Size              : chr  "Medium" "Medium" "Medium" "" ...
 $ Outlet_Location_Type     : chr  "Tier 1" "Tier 3" "Tier 1" "Tier 3" ...
 $ Outlet_Type              : chr  "Supermarket Type1" "Supermarket Type2" "Supermarket Type1"
 "Grocery Store" ...
 $ Item_Outlet_Sales        : num  3735 443 2097 732 995 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

**Performing Feature Engineering on the dataset**

**UNIT – IV**

**MODEL FIT USING R**

Regression Models- Linear and Logistic Model, Classification Models – Decision Tree, Naïve Bayes, SVM and Random Forest, Clustering Models – K Means and Hierarchical clustering

## 4.1. Regression Models:

Regression analysis is a statistical tool to estimate the relationship between two or more variables. There is always one response variable and one or more predictor variables. Regression analysis is widely used to fit the data accordingly and further, predicting the data for forecasting. It helps businesses and organizations to learn about the behavior of their product in the market using the dependent/response variable and independent/predictor variable. In this article, let us learn about different types of regression in R programming with the help of examples.

### Types of Regression in R:

There are mainly three types of Regression in R programming that is widely used. They are:

- Linear Regression
- Multiple Regression
- Logistic Regression

## 4.2. Linear Regression

The Linear Regression model is one of the widely used among three of the regression types. In linear regression, the relationship is estimated between two variables i.e., one response variable and one predictor variable. Linear regression produces a straight line on the graph. Mathematically

$$y = ax + b$$

Where,

- **x** indicates predictor or independent variable
- **y** indicates response or dependent variable
- **a** and **b** are coefficients

### Implementation in R

In R programming, **lm()** function is used to create linear regression model.

**Syntax:** lm(formula)
**Parameter:**

**formula:** represents the formula on which data has to be fitted To know about more optional parameters, use below command in console: help("lm")

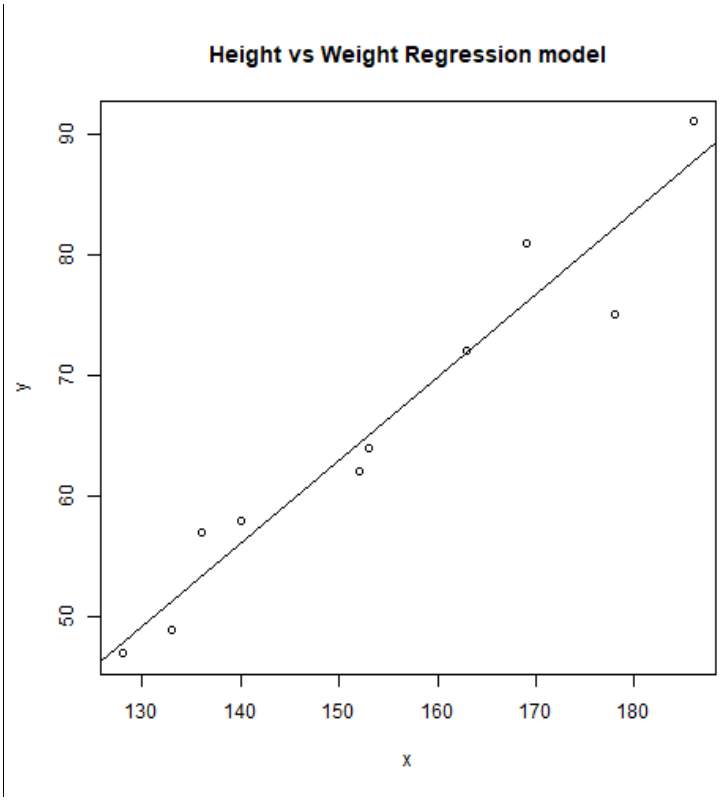
**Example:** In this example, let us plot the linear regression line on the graph and predict the weight-based using height.

```
# R program to illustrate

# Linear Regression


# Height vector

x <- c(153, 169, 140, 186, 128,

       136, 178, 163, 152, 133)


# Weight vector

y <- c(64, 81, 58, 91, 47, 57,

       75, 72, 62, 49)


# Create a linear regression model

model <- lm(y~x)


# Print regression model

print(model)


# Find the weight of a person

# With height 182

df <- data.frame(x = 182)

res <-  predict(model, df)

cat("\nPredicted value of a person

               with height = 182")

print(res)


# Output to be present as PNG file
```

```
png(file = "linearRegGFG.png")


# Plot

plot(x, y, main = "Height vs Weight

                  Regression model")

abline(lm(y~x))


# Save the file.

dev.off()
```

## Output:

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -39.7137       0.6847

Predicted value of a person with height = 182
      1
84.9098
```



### 4.3. Logistic Regression

Logistic Regression is another widely used regression analysis technique and predicts the value with a range. Moreover, it is used for predicting the values for categorical data. For example, Email is spam or non-spam, winner or loser, male or female, etc. Mathematically,

$$y = \frac{1}{1+e-2}$$

**where,**

- **y** represents response variable
- **z** represents equation of independent variables or features

**Implementation in R**

In R programming, **glm()** function is used to create a logistic regression model.

**Syntax:** glm(formula, data, family)

**Parameters:**

- **formula:** represents a formula on the basis of which model has to be fitted
- **data:** represents dataframe on which formula has to be applied
- **family:** represents the type of function to be used. "binomial" for logistic regression

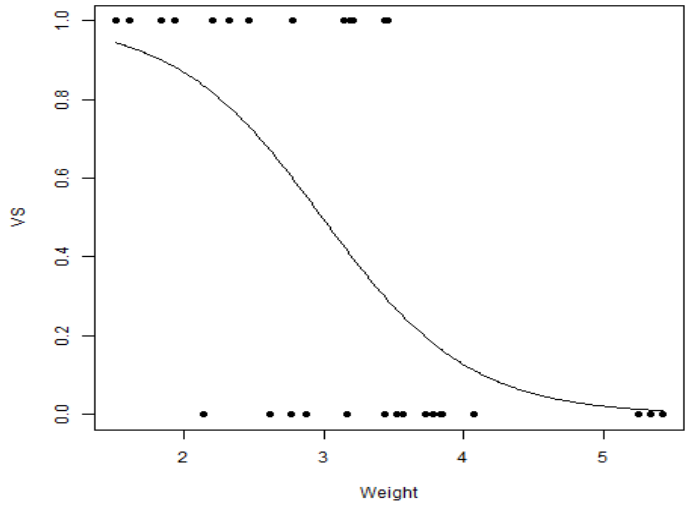
**Example:**

```
# R program to illustrate

# Logistic Regression


# Using mtcars dataset

# To create the logistic model

model <- glm(formula = vs ~ wt,

             family = binomial,

             data = mtcars)


# Creating a range of wt values
```

```
x <- seq(min(mtcars$wt),
             max(mtcars$wt),
             0.01)

# Predict using weight
y <- predict(model, list(wt = x),
             type = "response")

# Print model
print(model)

# Output to be present as PNG file
png(file = "LogRegGFG.png")

# Plot
plot(mtcars$wt, mtcars$vs, pch = 16,
     xlab = "Weight", ylab = "VS")
lines(x, y)

# Saving the file
dev.off()
```

**Output:**

```
Call:  glm(formula = vs ~ wt, family = binomial, data = mtcars)

Coefficients:
(Intercept)           wt
      5.715       -1.911

Degrees of Freedom: 31 Total (i.e. Null);  30 Residual
Null Deviance:      43.86
Residual Deviance: 31.37        AIC: 35.37
```




### 4.4. Classification in R Programming

R is a very dynamic and versatile programming language for data science. This article deals with classification in R. Generally classifiers in R are used to predict specific category related information like reviews or ratings such as good, best or worst. Various Classifiers are:

- Decision Trees
- Naive Bayes Classifiers
- K-NN Classifiers
- Support Vector Machines(SVM's)

#### 4.4.1.Decision Tree Classifier

It is basically is a graph to represent choices. The nodes or vertices in the graph represent an event and the edges of the graph represent the decision conditions. Its common use is in Machine Learning and Data Mining applications.
**Applications:**
Spam/Non-spam classification of email, predicting of a tumor is cancerous or not. Usually, a model is constructed with noted data also called training dataset. Then a set of validation data is used to verify and improve the model. R has packages that are used to create and visualize decision trees.
The R package "party" is used to create decision trees.
**Command:**

```
install.packages("party")


# Load the party package. It will automatically load other
# dependent packages.
```

```
library(party)

# Create the input data frame.
input.data <- readingSkills[c(1:105), ]

# Give the chart file a name.
png(file = "decision_tree.png")

# Create the tree.
  output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.dat)

# Plot the tree.
plot(output.tree)

# Save the file.
dev.off()
```

**Output:**

```
null device
          1
Loading required package: methods
Loading required package: grid
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
Loading required package: strucchange
Loading required package: zoo

Attaching package: 'zoo'

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

   as.Date, as.Date.numeric

Loading required package: sandwich
```

### 4.4.2. Naive Bayes Classifier

Naïve Bayes classification is a general classification method that uses a probability approach, hence also known as a probabilistic approach based on Bayes' theorem with the assumption of independence between features. The model is trained on training dataset to make predictions by **predict()** function.
**Formula:**

$$P(A|B)=P(B|A)\times P(A)P(B)$$

It is a sample method in machine learning methods but can be useful in some instances. The training is easy and fast that just requires considering each predictor in each class separately.

**Application:**
It is used generally in sentimental analysis.

```
library(caret)
## Warning: package 'caret' was built under R version 3.4.3
set.seed(7267166)
trainIndex = createDataPartition(mydata$prog, p = 0.7)$Resample1
train = mydata[trainIndex, ]
test = mydata[-trainIndex, ]

## check the balance
print(table(mydata$prog))
##
## academic    general vocational
## 105         45         50
print(table(train$prog))
```

**Output:**

```
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   academic    general vocational
##  0.5248227  0.2269504  0.2482270
##
## Conditional probabilities:
##             science
## Y               [, 1]     [, 2]
##   academic   54.21622 9.360761
##   general    52.18750 8.847954
##   vocational 47.31429 9.969871
##
##             socst
## Y               [, 1]      [, 2]
##   academic   56.58108  9.635845
##   general    51.12500  8.377196
##   vocational 44.82857 10.279865
```

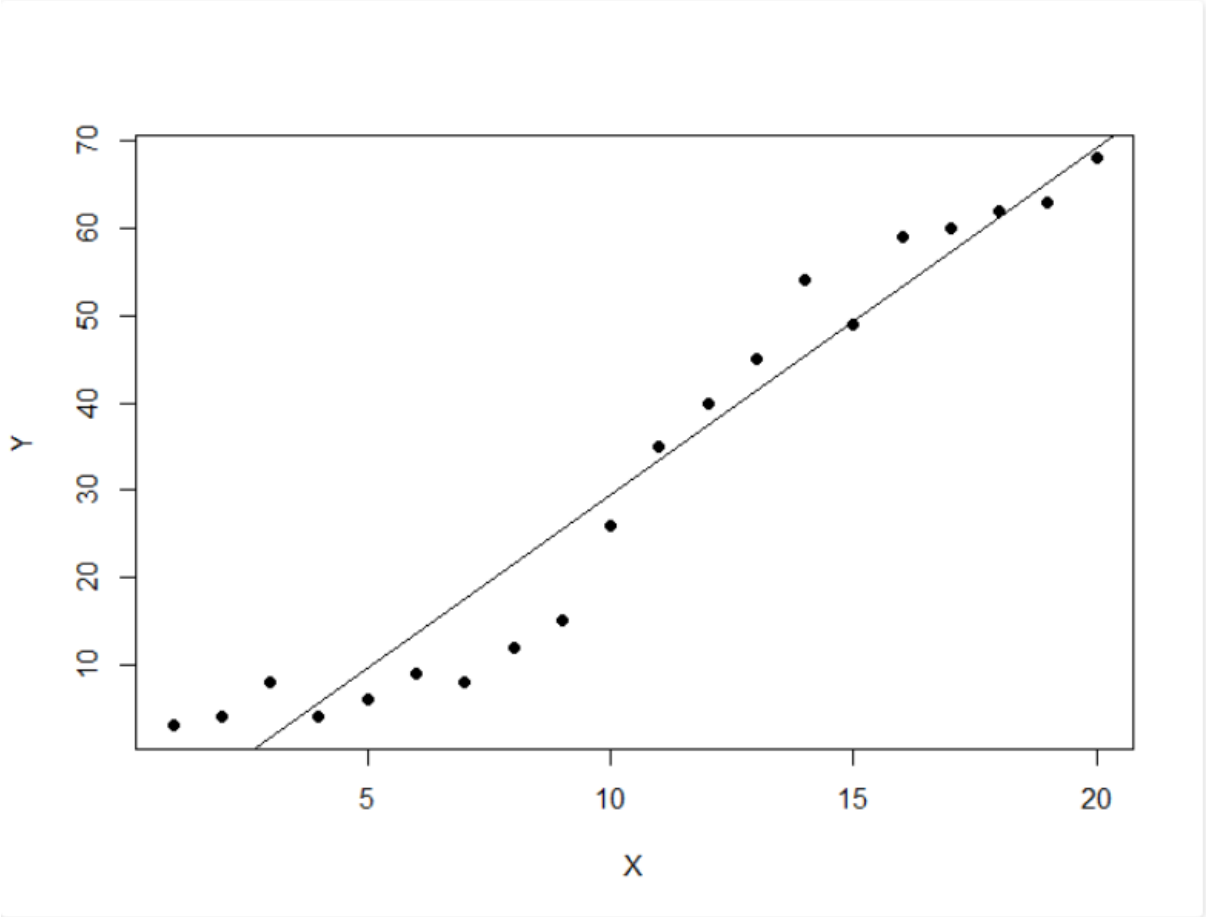
### 4.4.3. Support Vector Machines (SVM's)

A support vector machine (SVM) is a supervised binary machine learning algorithm that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text. Mainly SVM is used for text classification problems. It classifies the unseen data. It is widely used than Naive Bayes.SVM id usually a fast and dependable classification algorithm that performs very well with a limited amount of data.

**Applications:**
SVMs have a number of applications in several fields like Bioinformatics, to classify genes, etc.

```
# Load the data from the csv file
dataDirectory <- "D:/" # put your own folder here
data <- read.csv(paste(dataDirectory, 'regression.csv', sep =""), header =
TRUE)

# Plot the data
plot(data, pch = 16)

# Create a linear regression model
model <- lm(Y ~ X, data)

# Add the fitted line
abline(model)
```

**Output:**




### 4.4.5. Random Forest Approach in R Programming

Random Forest in R Programming is an ensemble of decision trees. It builds and combines multiple decision trees to get more accurate predictions. It's a non-linear classification algorithm. Each decision tree model is used when employed on its own. An error estimate of cases is made that is not used when constructing the tree. This is called an out of bag error estimate mentioned as a percentage.

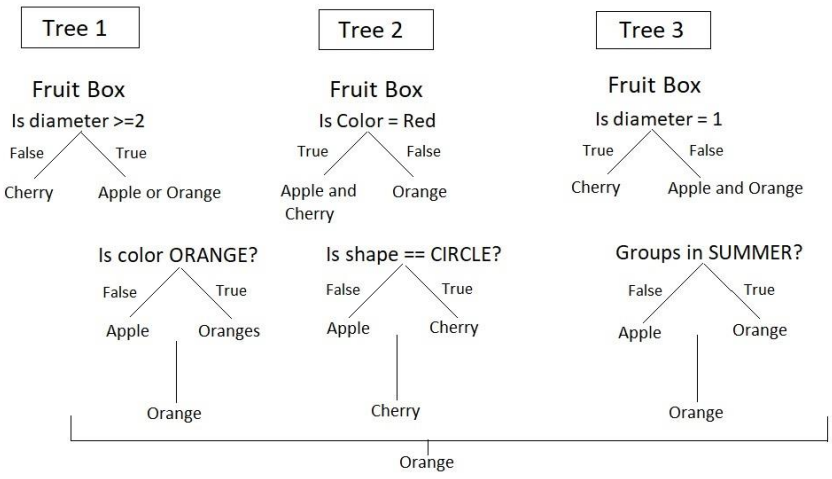
They are called **random** because they choose predictors randomly at a time of training. They are called **forest** because they take the output of multiple trees to make a decision. Random forest outperforms decision trees as a large number of uncorrelated trees(models) operating as a committee will always outperform the individual constituent models.

**Theory**

Random forest takes random samples from the observations, random initial variables(columns) and tries to build a model. Random forest algorithm is as follows:

- Draw a random bootstrap sample of size **n** (randomly choose **n** samples from training data).
- Grow a decision tree from bootstrap sample. At each node of tree, randomly select **d** features.
- Split the node using features(variables) that provide best split according to objective function. For instance, by maximizing the information gain.
- Repeat steps 1 to step 2, **k** times(k is the number of trees you want to create using subset of samples).
- Aggregate the prediction by each tree for a new data point to assign the class label by majority vote i.e pick the group selected by most number of trees and assign new data point to that group.

**Example:**
Consider a Fruit Box consisting of three fruits Apples, Oranges, and Cherries in training data i.e n = 3. We are predicting the fruit which is maximum in number in a fruit box. A random forest model using the training data with a number of trees, k = 3.




The model is judged using various features of data i.e diameter, color, shape, and groups. Among orange, cheery, and orange, orange is selected to be maximum in fruit box by random forest.

**The Dataset**

**`Iris`** dataset consists of 50 samples from each of 3 species of Iris(Iris setosa, Iris virginica, Iris versicolor) and a multivariate dataset introduced by British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems*. Four features were measured from each sample i.e length and width of the sepals and petals and based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

```
# Installing package
install.packages("caTools")       # For sampling the dataset
install.packages("randomForest")  # For implementing random forest
algorithm

# Loading package
library(caTools)
library(randomForest)

# Splitting data in train and test data
split <- sample.split(iris, SplitRatio = 0.7)
split

train <- subset(iris, split == "TRUE")
test <- subset(iris, split == "FALSE")

# Fitting Random Forest to the train dataset
set.seed(120)  # Setting seed
classifier_RF = randomForest(x = train[-5],
                             y = train$Species,
                             ntree = 500)

classifier_RF

# Predicting the Test set results
y_pred = predict(classifier_RF, newdata = test[-5])

# Confusion Matrix
confusion_mtx = table(test[, 5], y_pred)
confusion_mtx

# Plotting model
plot(classifier_RF)

# Importance plot
importance(classifier_RF)

# Variable importance plot
varImpPlot(classifier_RF)
```

**Output:**

```
> classifier_RF

Call:
 randomForest(x = train[-5], y = train$Species, ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.33%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         30          0         0  0.00000000
versicolor      0         29         1  0.03333333
virginica       0          2        28  0.06666667
```

### 4.5. Clustering in R Programming

**Clustering in R Programming Language** is an unsupervised learning technique in which the data set is partitioned into several groups called clusters based on their similarity. Several clusters of data are produced after the segmentation of data. All the objects in a cluster share common characteristics. During data mining and analysis, clustering is used to find similar datasets.

### Types of Clustering in R Programming

In R, there are numerous clustering algorithms to choose from. Here are a few of the most popular clustering techniques in R:

1. **K-means clustering:** it is a data-partitioning technique that seeks to assign each observation to the cluster with the closest mean after dividing the data into k clusters.

2. **Hierarchical clustering:** By repeatedly splitting or merging clusters according to their similarity, hierarchical clustering is a technique for creating a hierarchy of clusters.

3. **DBSCAN clustering:** it is a density-based technique that divides regions with lower densities and clusters together data points that are close to one another.

4. **Spectral clustering:** Spectral clustering is a technique that turns the clustering problem into a graph partitioning problem by using the eigenvectors of the similarity matrix.

5. **Fuzzy clustering:** Instead of allocating each data point to a single cluster, fuzzy clustering allows points to belong to numerous clusters with varying degrees of membership.

6. **Density-based clustering:** A class of techniques known as density-based clustering groups together data points based on density rather than distance.

7. **Ensemble clustering:** Ensemble clustering is a technique for enhancing clustering performance by combining several clustering methods or iterations of the same algorithm.
   Each kind of clustering technique has its own advantages and disadvantages and is appropriate for various kinds of data and clustering issues. The qualities of the data and the objectives of the research will determine which clustering technique is best.

**Applications of Clustering in R Programming Language**

- **Marketing:** In R programming, clustering is helpful for the marketing field. It helps in finding the market pattern and thus, helps in finding the likely buyers. Getting the interests of customers using clustering and showing the same product of their interest can increase the chance of buying the product.
- **Medical Science:** In the medical field, there is a new invention of medicines and treatments on a daily basis. Sometimes, new species are also found by researchers and scientists. Their category can be easily found by using the clustering algorithm based on their similarities.
- **Games:** A clustering algorithm can also be used to show the games to the user based on his interests.
- **Internet:** An user browses a lot of websites based on his interest. Browsing history can be aggregated to perform clustering on it and based on clustering results, the profile of the user is generated.

**Methods of Clustering**

There are 2 types of clustering in R programming:

- **Hard clustering:** In this type of clustering, the data point either belongs to the cluster totally or not and the data point is assigned to one cluster only. The algorithm used for hard clustering is k-means clustering.
- **Soft clustering:** In soft clustering, the probability or likelihood of a data point is assigned in the clusters rather than putting each data point in a cluster. Each data point exists in all the clusters with some probability. The algorithm used for soft clustering is the fuzzy clustering method or soft k-means.

**4.5.1. K-Means Clustering in R Programming language**

K-Means is an iterative hard clustering technique that uses an unsupervised learning algorithm. In this, total numbers of clusters are pre-defined by the user and based on the similarity of each data point, the data points are clustered. This algorithm also finds out the centroid of the cluster.

**Algorithm:**

- **Specify number of clusters (K):** Let us take an example of k =2 and 5 data points.
- **Randomly assign each data point to a cluster:** In the below example, the red and green color shows 2 clusters with their respective random data points assigned to them.

- **Calculate cluster centroids:** The cross mark represents the centroid of the corresponding cluster.
- **Re-allocate each data point to their nearest cluster centroid:** Green data point is assigned to the red cluster as it is near to the centroid of red cluster.
- **Re-figure cluster centroid**

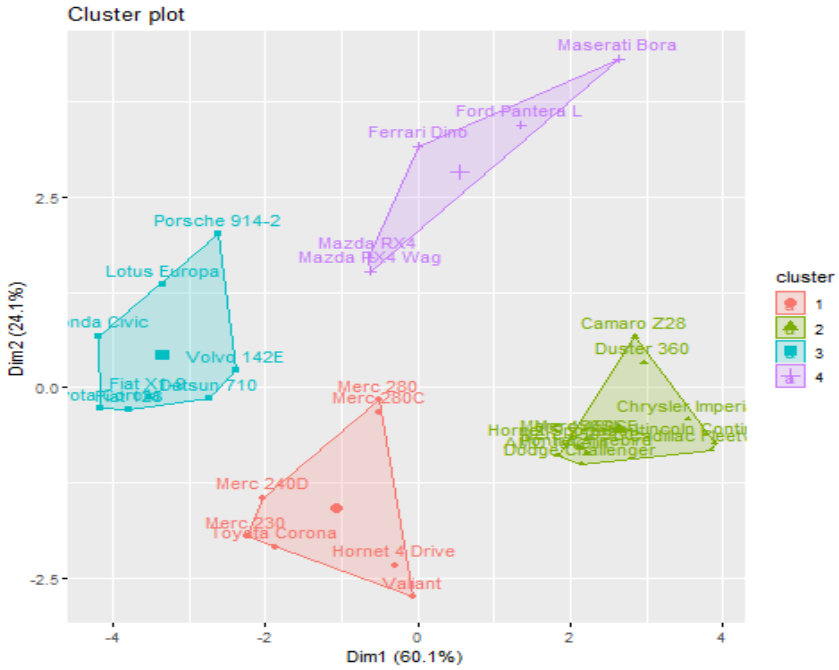
**Syntax:** kmeans(x, centers, nstart)

**where,**

- **x** represents numeric matrix or data frame object
- **centers** represents the **K** value or distinct cluster centers
- **nstart** represents number of random sets to be chosen

```
# Library required for fviz_cluster function
install.packages("factoextra")
library(factoextra)

# Loading dataset
df <- mtcars

# Omitting any NA values
df <- na.omit(df)

# Scaling dataset
df <- scale(df)

# output to be present as PNG file
png(file = "KMeansExample.png")

km <- kmeans(df, centers = 4, nstart = 25)

# Visualize the clusters
fviz_cluster(km, data = df)

# saving the file
dev.off()

# output to be present as PNG file
png(file = "KMeansExample2.png")

km <- kmeans(df, centers = 5, nstart = 25)

# Visualize the clusters
fviz_cluster(km, data = df)

# saving the file
dev.off()
```

- **Output:**
- When k = 4





Using the fviz_cluster function from the factoextra package, this code applies k-means clustering to the mtcars dataset using two different values of centers (4 and 5) and then saves the cluster visualizations as PNG files.

The clustering technique uses the information from the mtcars dataset, which includes details on various automobile models including the number of cylinders, horsepower, and miles per gallon. The scale function is used to scale the data to have a zero mean and unit variance, and the na.omit function is used to delete any rows with missing values.

Then, to improve the chance of discovering the global optimum, the means function is used to execute k-means clustering with 4 and 5 clusters. The fviz_cluster function, which plots the data points coloured by cluster membership and also displays the cluster centers, is then used to see the resulting cluster assignments.

The png and dev. off functions are then used to save the generated plots as PNG files.

To save the files to the correct location on your computer, you might need to modify the file paths or file names used in the png function.

### 4.5.2. Hierarchical Clustering in R Programming

**Hierarchical clustering in R Programming Language** is an unsupervised non-linear algorithm in which clusters are created such that they have a hierarchy (or a pre-determined ordering). For example, consider a family of up to three generations. A grandfather and mother have their children that become father and mother of their children. So, they all are grouped together to the same family i.e they form a hierarchy.

**R – Hierarchical Clustering**

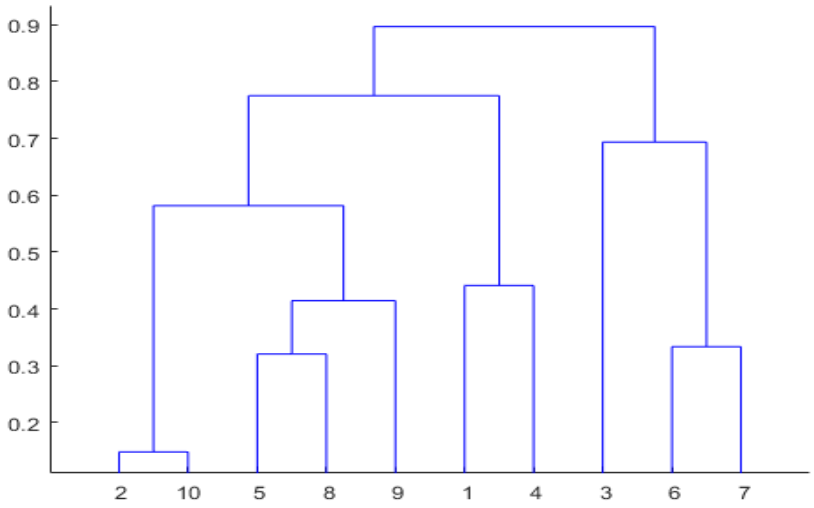
Hierarchical clustering is of two types:

- **Agglomerative Hierarchical clustering:** It starts at individual leaves and successfully merges clusters together. Its a Bottom-up approach.
- **Divisive Hierarchical clustering:** It starts at the root and recursively split the clusters. It's a top-down approach.

**Theory:**

In hierarchical clustering, Objects are categorized into a hierarchy similar to a tree-shaped structure which is used to interpret hierarchical clustering models. The algorithm is as follows:

- Make each data point in a single point cluster that forms N clusters.
- Take the two closest data points and make them one cluster that forms N-1 clusters.
- Take the two closest clusters and make them one cluster that forms N-2 clusters.
- Repeat steps 3 until there is only one cluster.




Dendrogram is a hierarchy of clusters in which distances are converted into heights. It clusters **n** units or objects each with **p** feature into smaller groups. Units in the same cluster are joined by a horizontal line. The leaves at the bottom represent individual units. It provides a visual representation of clusters.
**Thumb Rule:** Largest vertical distance which doesn't cut any horizontal line defines the optimal number of clusters.

**The Dataset**

**mtcars**(motor trend car road test) comprise fuel consumption, performance, and 10 aspects of automobile design for 32 automobiles. It comes pre-installed with dplyr package in R.

```
# Installing the package
install.packages("dplyr")

# Loading package
library(dplyr)

# Summary of dataset in package
head(mtcars)
```

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

**Performing Hierarchical clustering on Dataset**

Using Hierarchical Clustering algorithm on the dataset using **hclust()** which is pre-installed in stats package when R is installed.

```
# Finding distance matrix
distance_mat <- dist(mtcars, method = 'euclidean')
distance_mat

# Fitting Hierarchical clustering Model
# to training dataset
set.seed(240)  # Setting seed
Hierar_cl <- hclust(distance_mat, method = "average")
Hierar_cl

# Plotting dendrogram
plot(Hierar_cl)

# Choosing no. of clusters
# Cutting tree by height
abline(h = 110, col = "green")

# Cutting tree by no. of clusters
fit <- cutree(Hierar_cl, k = 3 )
fit

table(fit)
rect.hclust(Hierar_cl, k = 3, border = "green")
```

**Output:**

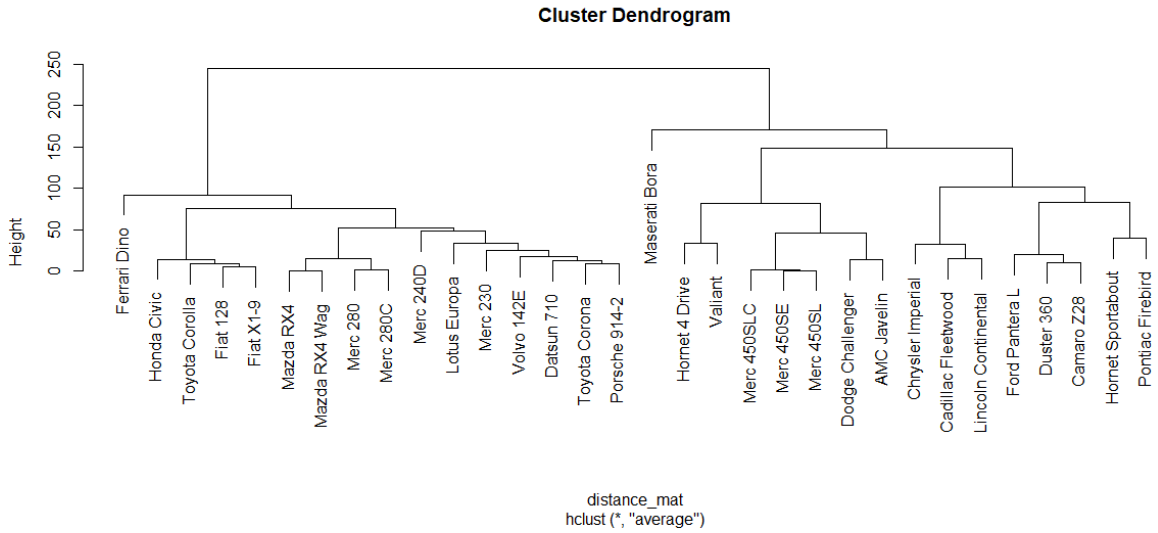
- **Distance matrix:**

```
> distance_mat
                     Mazda RX4 Mazda RX4 Wag  Datsun 710 Hornet 4 Drive Hornet Sportabout
Mazda RX4 Wag        0.6153251
Datsun 710          54.9086059    54.8915169
Hornet 4 Drive      98.1125212    98.0958939 150.9935191
Hornet Sportabout  210.3374396   210.3358546 265.0831615    121.0297564
Valiant             65.4717710    65.4392224 117.7547018     33.5508692       152.1241352
Duster 360         241.4076490   241.4088680 294.4790230    169.4299647        70.1767262
Merc 240D           50.1532711    50.1146059  49.6584796    121.2739722       241.5069657
Merc 230            25.4683117    25.3284509  33.1803843    118.2433145       233.4924012
Merc 280            15.3641921    15.2956865  66.9363534     91.4224033       199.3344960
Merc 280C           15.6724727    15.5837744  67.0261397     91.4612914       199.3406564
Merc 450SE         135.4307018   135.4254826 189.1954941     72.4964325        84.3888482
Merc 450SL         135.4014424   135.3960351 189.1631745     72.4313532        84.3683999
Merc 450SLC        135.4794674   135.4723157 189.2345426     72.5718466        84.4332423
Cadillac Fleetwood 326.3395903   326.3355070 381.0926242    234.4403876       116.2804201
Lincoln Continental 318.0469808  318.0429333 372.8012090    227.9726091       108.0624299
Chrysler Imperial  304.7203408   304.7169175 359.3014906    218.1548299        97.2049146
Fiat 128            93.2679950    93.2530993  40.9933763    184.9689734       302.0377212
Honda Civic        102.8307567   102.8238713  52.7704607    191.5518700       310.0324645
Toyota Corolla     100.6040368   100.5887588  47.6535017    192.6714187       309.5581776
Toyota Corona       42.3075233    42.2659224  12.9654743    138.5304725       252.3331988
Dodge Challenger   163.1150750   163.1134210 217.7795805     72.4403915        48.9838851
AMC Javelin        149.6047203   149.6014522 204.3188913     61.3601899        61.4274240
Camaro Z28         233.2228758   233.2248748 286.0049209    163.6632641        70.9665308
Pontiac Firebird   248.6780270   248.6762035 303.3583889    156.2240346        40.0052475
Fiat X1-9           92.5048389    92.4940020  39.8815148    184.4471198       301.5669483
Porsche 914-2       44.4033659    44.4073589  13.1357109    139.1579524       254.1452553
Lotus Europa        65.7328377    65.7362635  25.0948550    163.2367437       272.3582423
Ford Pantera L     245.4247064   245.4293785 297.2940489    180.1140339        89.5934049
Ferrari Dino        66.7661029    66.7764167  90.2415509    130.5523007       215.0673853
Maserati Bora      265.6454248   265.6491465 309.7718171    229.3419352       170.7094473
Volvo 142E          39.1894029    39.1626037  20.6939436    137.0363299       248.0063378
```

- The values are shown as per the distance matrix calculation with the method as euclidean.
- **Model Hierar_cl:**

```
> Hierar_cl

Call:
hclust(d = distance_mat, method = "average")

Cluster method   : average
Distance         : euclidean
Number of objects: 32
```

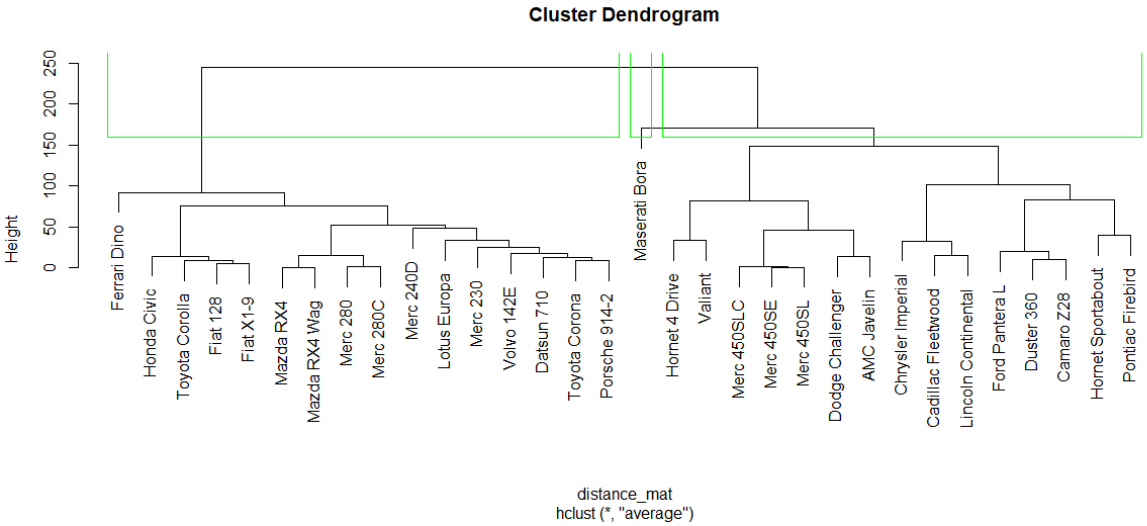
- In the model, the cluster method is average, distance is euclidean and no. of objects are 32.
- **Plot dendrogram:**




- The plot dendrogram is shown with x-axis as distance matrix and y-axis as height.
- **Cutted tree:**

```
> fit
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive
                  1                   1                   1                   2
  Hornet Sportabout             Valiant             Duster 360          Merc 240D
                  2                   2                   2                   1
           Merc 230            Merc 280            Merc 280C           Merc 450SE
                  1                   1                   1                   2
          Merc 450SL          Merc 450SLC  Cadillac Fleetwood Lincoln Continental
                  2                   2                   2                   2
  Chrysler Imperial            Fiat 128          Honda Civic      Toyota Corolla
                  2                   1                   1                   1
      Toyota Corona      Dodge Challenger          AMC Javelin          Camaro Z28
                  1                   2                   2                   2
   Pontiac Firebird            Fiat X1-9       Porsche 914-2        Lotus Europa
                  2                   1                   1                   1
     Ford Pantera L         Ferrari Dino       Maserati Bora          Volvo 142E
                  2                   1                   3                   1
```

- So, Tree is cut where k = 3 and each category represents its number of clusters.
- **Plotting dendrogram after cutting:**




- The plot denotes dendrogram after being cut. The green lines show the number of clusters as per the thumb rule.

## UNIT – V

## VISUALIZATION

Data visualization: Box plot, histogram, scatter plot, heat map – Working with Tableau – Outlier detection - Data Balancing

**Data visualization:**

**Data visualization** is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana. The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set. This article discusses the use of R for data visualization.

R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.

Consider the following air quality data set for visualization in R:

| Ozone | Solar R. | Wind | Temp | Month | Day |
|---|---|---|---|---|---|
| 41 | 190 | 7.4 | 67 | 5 | 1 |
| 36 | 118 | 8.0 | 72 | 5 | 2 |
| 12 | 149 | 12.6 | 74 | 5 | 3 |
| 18 | 313 | 11.5 | 62 | 5 | 4 |
| NA | NA | 14.3 | 56 | 5 | 5 |
| 28 | NA | 14.9 | 66 | 5 | 6 |

**Types of Data Visualizations**

Some of the various types of visualizations offered by R are:

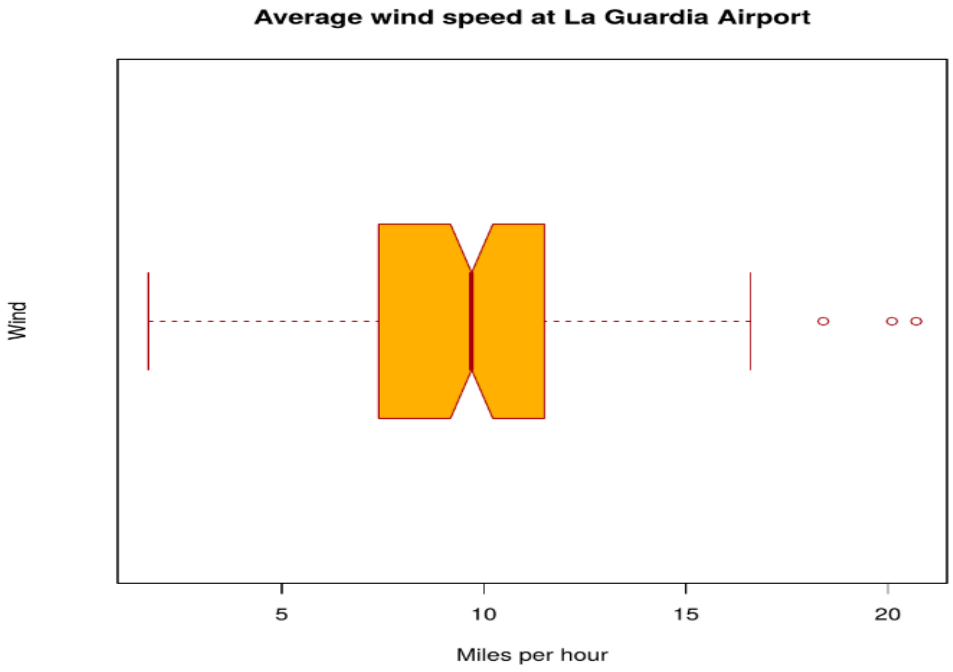
**Box Plot**

The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.

**Example:**

```
# Box plot for average wind speed

data(airquality)


boxplot(airquality$Wind, main = "Average wind speed\

at La Guardia Airport",

        xlab = "Miles per hour", ylab = "Wind",

        col = "orange", border = "brown",

        horizontal = TRUE, notch = TRUE)
```

**Output:**

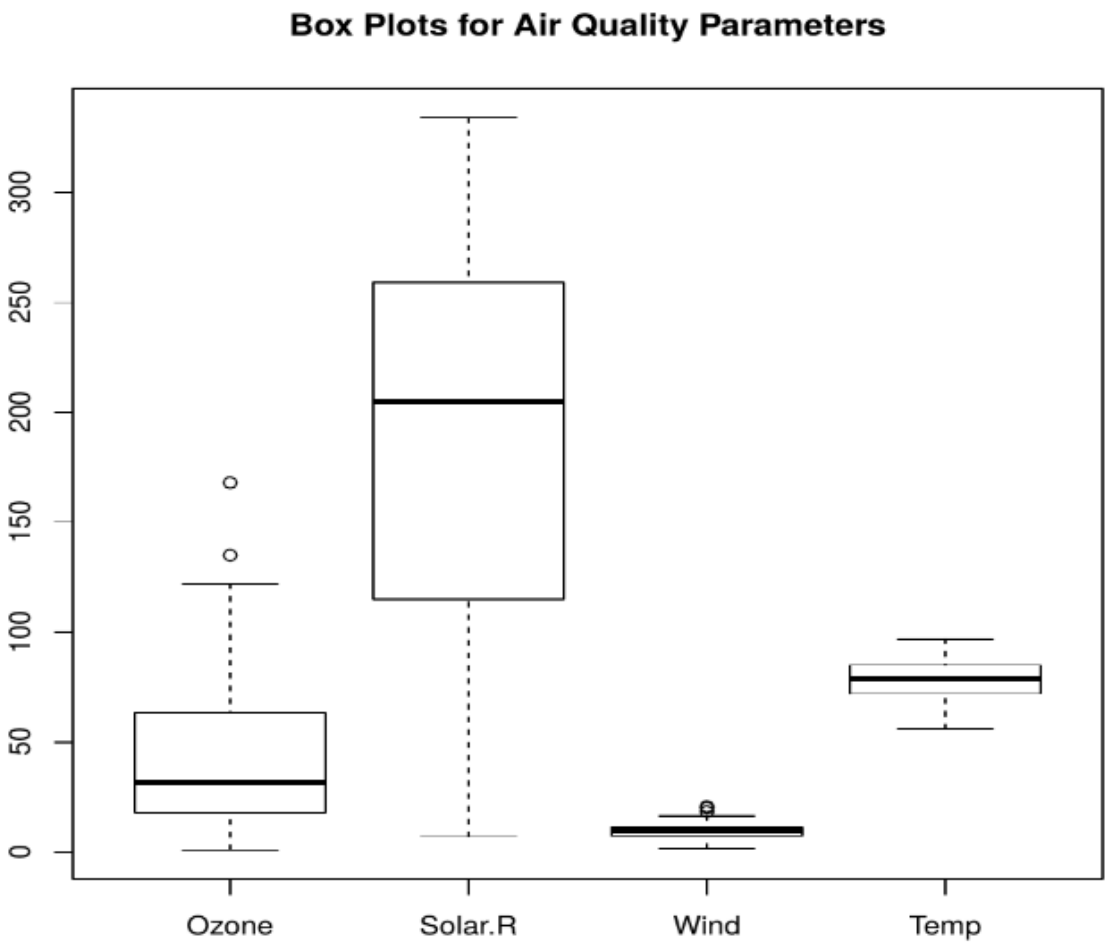



Multiple box plots can also be generated at once through the following code:

**Example:**

```
# Multiple Box plots, each representing
# an Air Quality Parameter
boxplot(airquality[, 0:4],
        main ='Box Plots for Air Quality Parameters')
```

**Output:**




**Box Plots are used for:**

- To give a comprehensive statistical description of the data through a visual cue.
- To identify the outlier points that do not lie in the inter-quartile range of data.

**Scatter Plot**

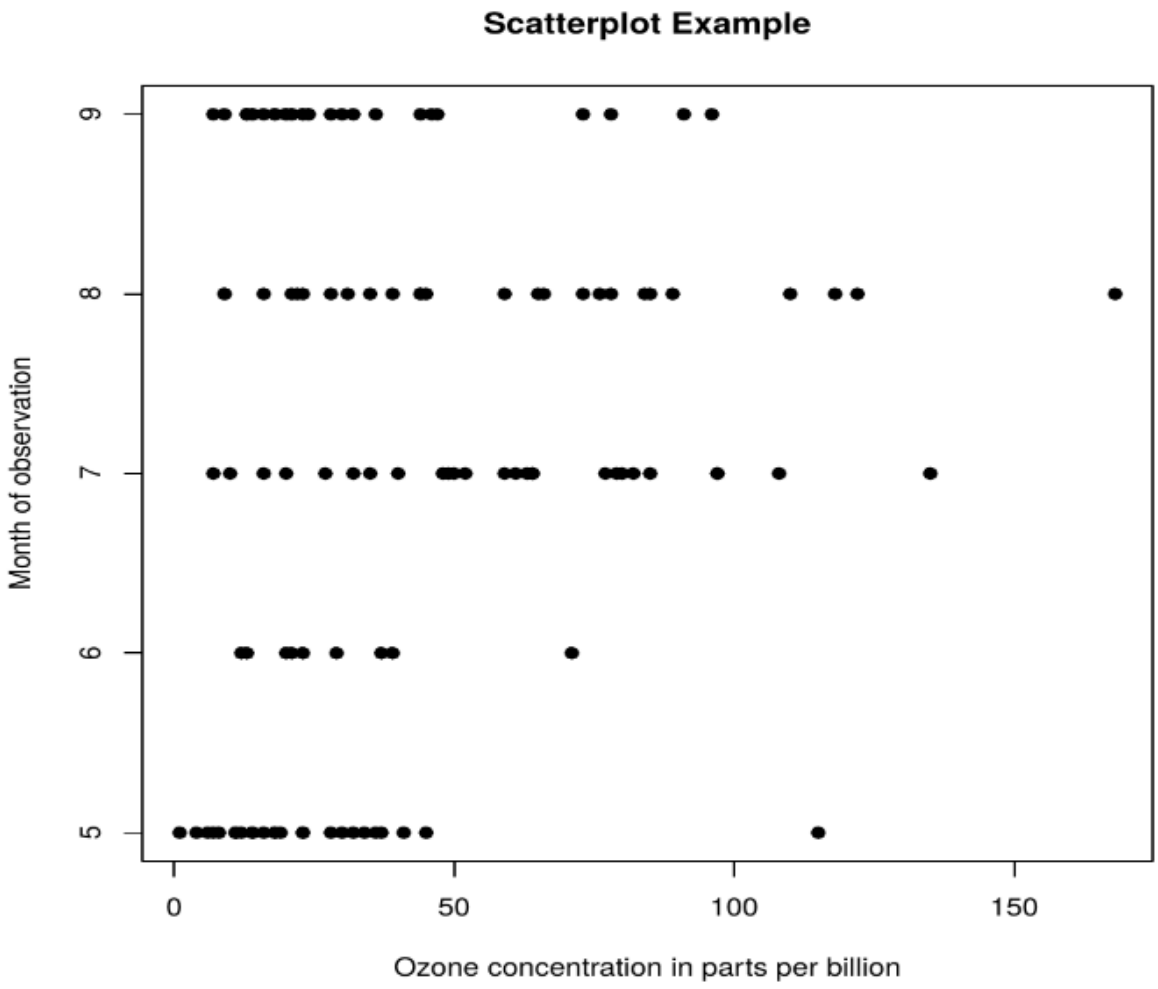
A scatter plot is composed of many points on a Cartesian plane. Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.

**Example:**

```
# Scatter plot for Ozone Concentration per month

data(airquality)


plot(airquality$Ozone, airquality$Month,
```

```
 main ="Scatterplot Example",
xlab ="Ozone Concentration in parts per billion",
ylab =" Month of observation ", pch = 19)
```

**Output:**




**Scatter Plots are used in the following scenarios:**

- To show whether an association exists between bivariate data.
- To measure the strength and direction of such a relationship.

**Heat Map**

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. heatmap() function is used to plot heatmap.

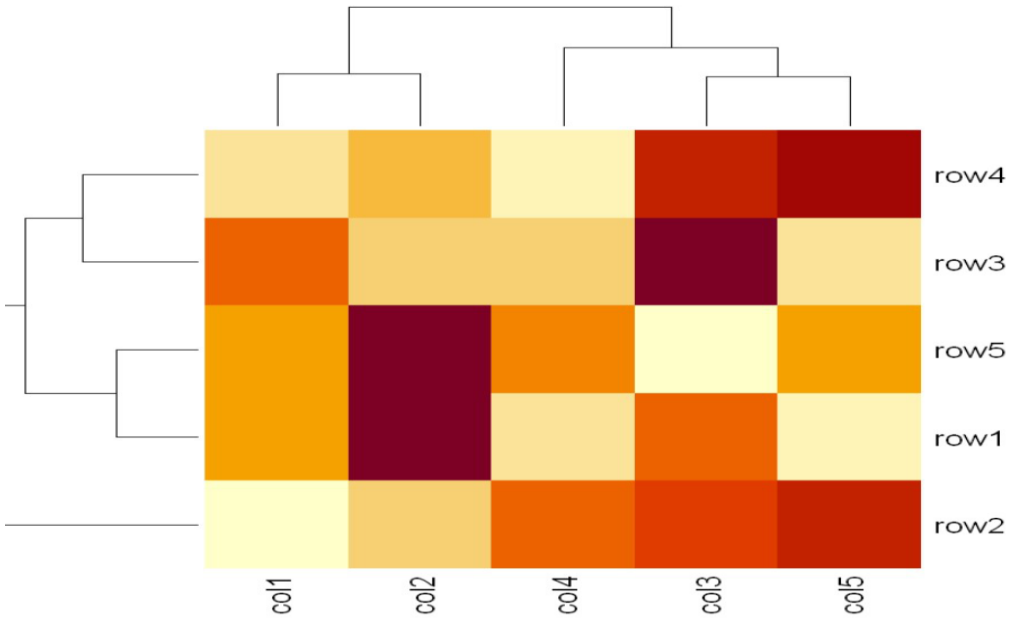
**Syntax:** heatmap(data)

**Parameters:** data: It represent matrix data, such as values of rows and columns

**Return:** This function draws a heatmap.

```
# Set seed for reproducibility

# set.seed(110)


# Create example data

data <- matrix(rnorm(50, 0, 5), nrow = 5, ncol = 5)


# Column names

colnames(data) <- paste0("col", 1:5)

rownames(data) <- paste0("row", 1:5)


# Draw a heatmap

heatmap(data)
```

**Output:**




**Advantages of Data Visualization in R:**

R has the following advantages over other tools for data visualization:

- R offers a broad collection of visualization libraries along with extensive online guidance on their usage.
- R also offers data visualization in the form of 3D models and multipanel charts.
- Through R, we can easily customize our data visualization by changing axes, fonts, legends, annotations, and labels.

**Disadvantages of Data Visualization in R:**

R also has the following disadvantages:

- R is only preferred for data visualization when done on an individual standalone server.
- Data visualization using R is slow for large amounts of data as compared to other counterparts.

**Application Areas:**

- Presenting analytical conclusions of the data to the non-analysts departments of your company.
- Health monitoring devices use data visualization to track any anomaly in blood pressure, cholesterol and others.
- To discover repeating patterns and trends in consumer and marketing data.
- Meteorologists use data visualization for assessing prevalent weather changes throughout the world.
- Real-time maps and geo-positioning systems use visualization for traffic monitoring and estimating travel time.

**Working with Tableau:**

Tableau is very famous as it can take in data and produce the required data visualization output in a very short time. Basically, it can elevate your data into insights that can be used to drive your action in the future. And Tableau can do all this while providing the highest level of security with a guarantee to handle security issues as soon as they arise or are found by users.

Tableau also allows you to prepare, clean, and format data of all types and ranges and then create data visualizations to obtain actionable insights that can be shared with other users. You can use data queries to obtain insights from your visualizations and also manage metadata using Tableau. In fact, it is a lifesaver for many people in Business Intelligence as it allows you to handle data without having great technical knowledge. So you can use Tableau as an individual data analyst or at a large scale for your business team and organization. In fact, there are many organizations using Tableau such as Amazon, Lenovo, Walmart, Accenture, etc. There are different Tableau products that are aimed at different types of users, whether they be individuals or organizations. So let’s see these in detail now.

Tableau is a data visualization tool and it allows connecting with a large range of data sources, creating interactive visualizations, and providing features to share work with other team members. It is also used by data analysts and data scientists to explore data and create visualizations that communicate understandings to others. It has an interface that allows drag-and-drop data areas to create charts, graphs, and visualizations to analyze the data which is

easier to use for beginners also. It is used by businesses like medicine, technology, e-commerce, etc, to analyze data and make data-driven judgments.

**Connecting Data Sources:**

- To **create a visualization in Tableau**, we need to connect with the data source, which includes spreadsheets, cloud services, and databases.
- To connect to a data source, need to provide Tableau with the necessary information, login certifications, and the place of the data source. after we connect to a data source, we are able to see the available data fields and start creating visualizations in Tableau for the analysis of data.

**Advantages of Tableau**

Let's check out some of the advantages of Tableau:

**1. Create great visualizations**

Of course, the first advantage of a data visualization tool is that you can create wonderful and detailed data visualizations using data that initially wasn't very ordered. You can use Tableau Prep to shape, clean, and combine the data into desired forms so that it can be used for creating data charts, dashboards, visualizations, etc.

**2. Obtain detailed insights**

You can obtain detailed and unexpected insights from the data using Tableau. You can explore the data from different angles to see if any patterns emerge or you can even ask open-ended questions from the data and perform various comparisons to obtain unexpected insights. This effect is heightened even more when you are using real-time data as it changes your viewpoint continuously.

**3. User-friendly Approach**

Tableau is created for people who don't have detailed technical skills or much coding experience and so its user-friendly approach is its greatest strength. You can create detailed data visualizations from Tableau without having many technical skills as most of its features use a drag-and-drop approach to put the correct parameters in the rows and columns to create visualizations. This is so simple and intuitive that even a layman can manage it.

**4. Support for Different Data Sources**

Tableau can connect to various data sources, data warehouses, and files that contain disparate data and exist in different kinds of storage mediums. Tableau can access data from the cloud, data that is available in spreadsheets, big data, non-relational data, etc. Tableau has the capacity to manage data from all these different data sources and blend these different types of data to create complex and detailed data visualizations that are an asset to IT companies.

## What are outliers?

Data points far from the dataset's other points are considered outliers. This refers to the data values dispersed among other data values and upsetting the dataset's general distribution. Outlier detection is a statistical approach used to find outliers in datasets. Measurement errors, incorrect data entry, or really anomalous data values are just a few of the causes of outliers.

**Effects of an outlier on the model:**

- The format of the data appears to be skewed.
- Modifies the mean, variance, and other statistical characteristics of the data's overall distribution.
- Leads to the model's accuracy level being biased.

**Steps involving Outlier detection:**

**Step 1:** In this step, we will be, by default creating the data containing the outliner inside it using the rnorm() function and generating 500 different data points. Further, we will be adding 10 random outliers to this data.

```
data <- rnorm(500)
data[1:10] <- c(46,9,15,-90,
         42,50,-82,74,61,-32)
```

**Step 2:** In this step, we will be analyzing the outliner in the provided data using the boxplot, which will be plotting a barplot, and we will be able to analyze the outlier in the data. As said when reviewing a box plot, an outlier is defined as a data point that is located outside the whiskers of the box plot.

**boxplot() function:**

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

**Syntax:** boxplot(x, data, notch, varwidth, names, main)

**Parameters:**

**x:** This parameter sets as a vector or a formula.

**data:** This parameter sets the data frame.

**notch:** This parameter is the label for horizontal axis.

**varwidth:** This parameter is a logical value. Set as true to draw width of the box proportionate to the sample size.

**main:** This parameter is the title of the chart.

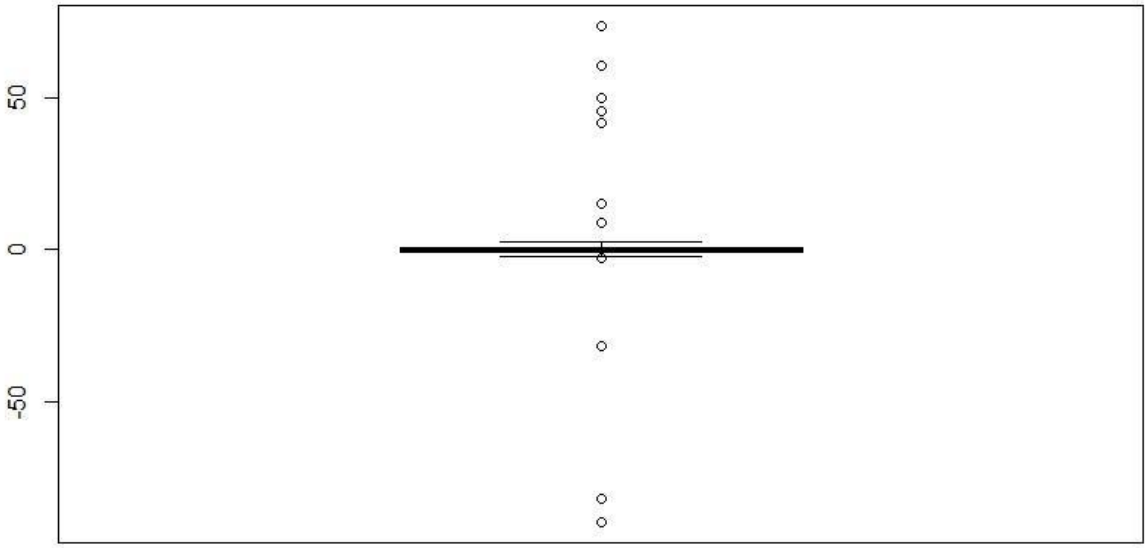
**names:** This parameter are the group labels that will be showed under each boxplot.

data <- rnorm(500)

data[1:10] <- c(46,9,15,-90,

42,50,-82,74,61,-32)

boxplot(data)



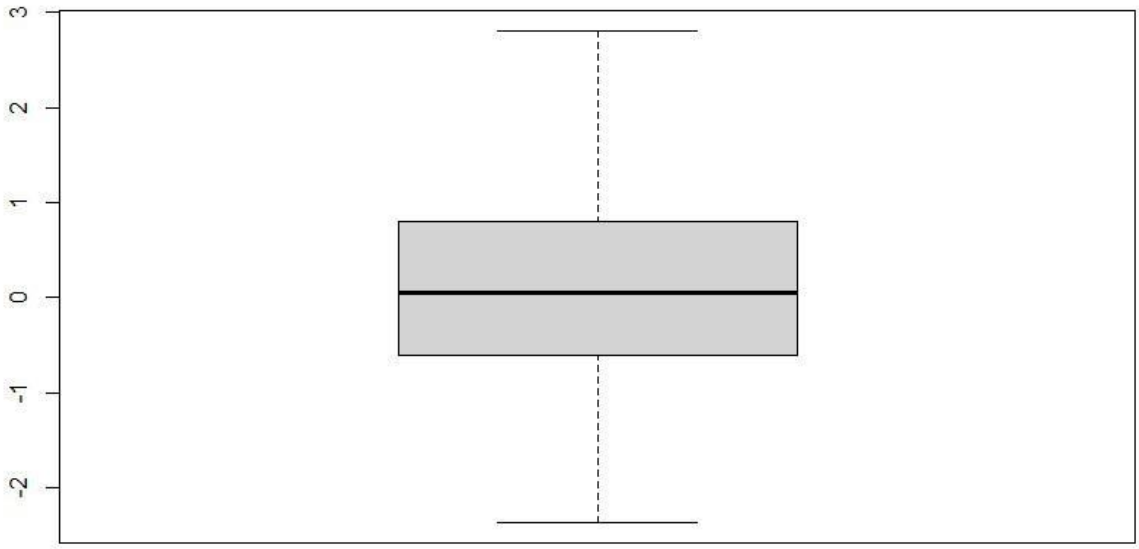

**Step 3: In this step, we will remove the outlier of the provided data boxplot.stats() function in R; the same illustration is shown in the below code.**

**data <- data[!data %in% boxplot.stats(data)$out]**

**Step 4: In this step, we will just verify if the outliner has been removed from the data simply by plotting the boxplot as done in step 2 and verifying it accordingly.**

```
data <- rnorm(500)
data[1:10] <- c(46,9,15,-90,42,50,-82,74,61,-32)
data <- data[!data %in% boxplot.stats(data)$out]
boxplot(data)
```

**Output:**

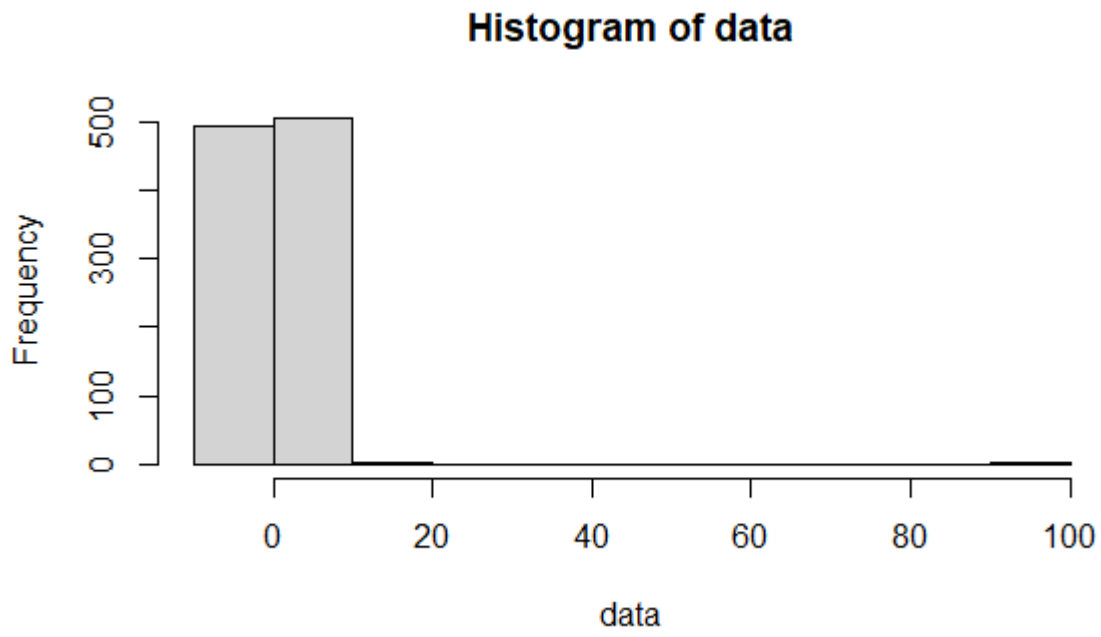



**Histogram function:**

Here we created a histogram by using the Histogram function in the R programming language to detect outliers.

```
# Generate a random dataset with outliers

set.seed(123)

data <- c(rnorm(1000), 10, 15, 12, 100)


# Create a histogram

hist(data)
```

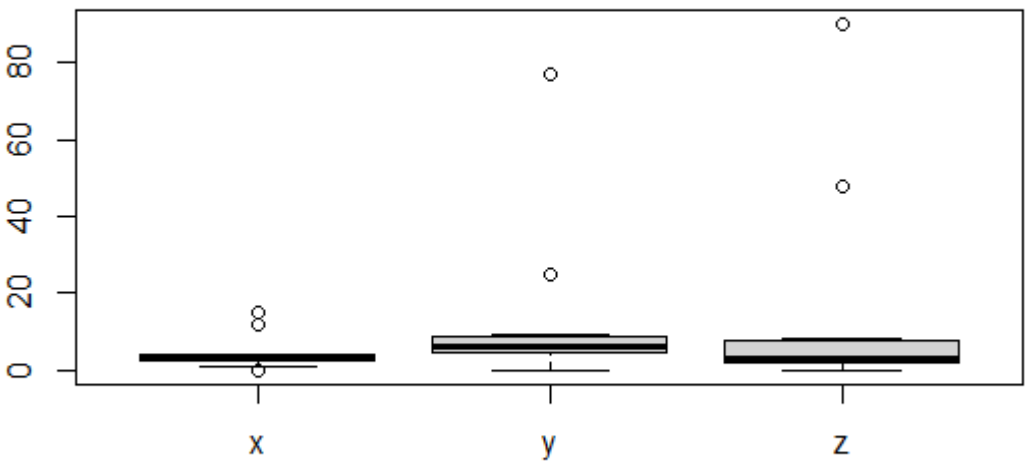
**Output:**




**Detect and remove outliers from multiple columns in the R dataframe:**

To detect and remove outliers from a data frame, we use the Interquartile range (IQR) method. This method uses the first and third quantile values to determine whether an observation is an outlier to not. If an observation is 1.5 times the interquartile range greater than the third quartile or 1.5 times the interquartile range less than the first quartile it is considered an outlier.

```
# create sample data frame

sample_data <- data.frame(x=c(1, 2, 3, 4, 3, 12, 3, 4, 4, 15, 0),

                          y=c(4, 3, 25, 7, 8, 5, 9, 77, 6, 5, 0),

                          z=c(1, 3, 2, 90, 8, 7, 0, 48, 7, 2, 3))

print("Display original dataframe")

print(sample_data)

boxplot(sample_data)
```


0
20
40
60
80
x
y
z