**Experiment 1**

Big Data refers to extremely large datasets that are complex and cannot be processed using traditional data processing tools. The three primary characteristics of Big Data are:

1. **Volume**: Huge amounts of data generated daily from various sources, such as social media, IoT devices, transactions, etc.
2. **Velocity**: The speed at which data is generated and processed to meet demands.
3. **Variety**: Data comes in various formats such as structured, semi-structured, and unstructured (e.g., text, images, videos).

**Use of Big Data:**  
Big Data is widely used across industries to derive insights, improve decision-making, enhance customer experiences, and predict future trends. Examples include fraud detection, personalized marketing, and predictive maintenance.

**Pandas**

Pandas is a powerful Python library for data manipulation and analysis. It provides data structures like **DataFrame** and **Series** for organizing and analyzing data efficiently.

* **Key Features:**
  + Easy handling of missing data.
  + Data wrangling and cleaning.
  + Support for reading and writing data from multiple file formats (CSV, Excel, SQL, etc.).

**Matplotlib**

Matplotlib is a Python plotting library that allows the creation of static, animated, and interactive visualizations.

* **Key Features:**
  + Customizable plots like line charts, bar graphs, histograms, and scatter plots.
  + High control over graphical elements such as axes, labels, and styles.

Seaborn is a Python library built on Matplotlib for creating visually appealing and informative statistical graphics.

* **Key Features:**
  + High-level interfaces for drawing attractive and informative statistical plots.
  + Automatic handling of pandas DataFrames.
  + Built-in themes for customization.

The histplot function in Seaborn is used to create histograms to visualize the distribution of a dataset.

* **Key Features:**
  + Combines histogram computation with kernel density estimation (optional).
  + Supports categorical and numerical data.

**Experiment 2**

**Data Visualization** is the process of representing data in graphical or visual formats, such as charts, graphs, and maps, to make complex information easier to understand. It helps in identifying patterns, trends, and outliers, enabling better decision-making. Tools like R, Python, Tableau, and Power BI are widely used for this purpose.

**Data Analysis** involves systematically applying statistical and logical techniques to describe, illustrate, and evaluate data. It aims to extract meaningful insights from raw data to inform decisions. Data analysis can include descriptive, inferential, predictive, and prescriptive methods.

**Libraries Used for Text Analysis and Visualization in R**

1. **tm (Text Mining):**  
   This library is essential for preprocessing text data, cleaning it, and preparing it for analysis. It supports operations like removing stop words, tokenization, and creating corpora.  
   *Example*: Creating a text corpus from a collection of documents.
2. **SnowballC (Stemming):**  
   Used for reducing words to their root form, ensuring consistency in text analysis. For example, "running," "runs," and "run" are all reduced to "run."  
   *Why*: Stemming helps in grouping similar words during analysis.
3. **wordcloud (Word Cloud Visualization):**  
   This library is used for generating word clouds, which represent the most frequent words in a dataset in a visually appealing manner.  
   *Why*: Useful for quickly understanding text data and identifying dominant terms.
4. **ggplot2 (Visualization):**  
   A robust library for creating layered, customizable visualizations like scatter plots, bar charts, and histograms.  
   *Why*: It provides flexibility and control over the aesthetic mapping of graphs.

**Key Concepts**

**1. Corpus**

A **corpus** is a collection of text documents that are analyzed collectively. In text mining, a corpus represents the dataset for processing, such as emails, tweets, or articles. It is the starting point for tasks like cleaning, stemming, and analysis.

*Example*: A corpus of customer reviews collected from an e-commerce platform

**2. Term Document Matrix (TDM)**

A Term Document Matrix is a matrix representation where rows represent unique terms (words) in the corpus, and columns represent individual documents. The values indicate the frequency of terms in each document.

*Purpose*: It is used for analyzing patterns, identifying frequent terms, and enabling machine learning tasks.

**3. Word Frequencies**

Word frequencies refer to the number of times a word appears in a given text or document. They are crucial for understanding the importance of terms in text analysis.

*Usage*: High-frequency words can indicate themes or topics within the corpus, while low-frequency words might hold unique insights.

**4. Term Frequency Matrix (TFM)**

A Term Frequency Matrix is similar to a TDM but focuses on the frequency of terms across the entire dataset rather than their distribution across individual documents.

*Purpose*: TFM helps in identifying the importance of terms in a corpus, often used for weighting methods like TF-IDF (Term Frequency-Inverse Document Frequency).

**Experiment 3**

Apache Spark is an open-source, distributed computing framework designed for big data processing. It provides in-memory processing capabilities, making it faster than traditional frameworks like Hadoop MapReduce. Spark supports various tasks such as batch processing, real-time streaming, machine learning, and graph processing.

**Why Use Apache Spark?**

1. **Speed:** It processes data much faster due to in-memory computation and optimized execution.
2. **Scalability:** Can handle petabytes of data across distributed systems.
3. **Flexibility:** Supports multiple programming languages like Python, Java, Scala, and R.
4. **Rich Ecosystem:** Integrates with libraries for machine learning (MLlib), streaming (Spark Streaming), and SQL (Spark SQL).

### **Explanation of Commands**

#### !apt-get install openjdk-8-jdk-headless -qq

* This command installs OpenJDK 8, a Java Development Kit (JDK) that Apache Spark requires to run, as Spark is built on Java Virtual Machine (JVM).
* **Why**: Ensures the necessary runtime environment for Spark applications.

#### !pip install -q pyspark

* Installs **PySpark**, the Python API for Apache Spark, enabling developers to use Python for big data processing.
* **Why**: Provides Python bindings to leverage Spark’s distributed computing capabilities.

### **Code Walkthrough**

1. **Importing SparkSession:**

python

Copy code

from pyspark.sql import SparkSession

* + SparkSession is the entry point for building and executing a Spark application. It manages the SparkContext and configurations.

1. **Initializing the Spark Session:**

python

Copy code

spark = SparkSession.builder.master("local[\*]").appName("WordCount").getOrCreate()

* + **master("local[\*]"):** Runs the application locally, utilizing all available CPU cores ([\*]).
  + **appName("WordCount"):** Names the Spark application.
  + **getOrCreate():** Returns an existing session if available or creates a new one.

1. **Creating a Text File:**

python

Copy code

text = ["Bhavani Rajpurohit", "Roll number", "2213688"]

with open("example.txt", "w") as f:

for line in text:

f.write(line + "\n")

* + A small text file named example.txt is created for the word count program.

1. **Reading the Text File as an RDD:**

python

Copy code

text\_file = spark.read.text("example.txt").rdd

* + Reads the file into an **RDD** (Resilient Distributed Dataset), Spark’s fundamental data structure for parallel processing.

1. **Performing the Word Count:**

python

Copy code

words = text\_file.flatMap(lambda line: line.value.split(" "))

word\_counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

* + **flatMap:** Splits each line into words and flattens them into a single list.
  + **map:** Maps each word to a key-value pair, (word, 1).
  + **reduceByKey:** Aggregates the counts for each unique word by summing the values.

1. **Collecting and Printing Results:**

python

Copy code

word\_counts = word\_counts.collect()

for word, count in word\_counts:

print(f"{word}: {count}")

* + **collect():** Gathers the results from the distributed RDD to the driver node for display.

1. **Stopping the Spark Session:**

python

Copy code

spark.stop()

* + Gracefully stops the Spark session, freeing up resources.

**Experiment 4**

**Objective**

1. **To load a CSV file into a PySpark DataFrame:**  
   PySpark DataFrames provide a distributed collection of data organized into named columns. This experiment demonstrates how to load data from a CSV file and analyze it using PySpark.
2. **To create and query a Hive table using PySpark:**  
   Hive tables are used to store structured data in Apache Hive, a data warehouse system built on top of Hadoop. PySpark provides seamless integration with Hive for creating and querying tables.

**What is a Hive Table?**

A **Hive Table** is a logical structure to store data in the Apache Hive framework, similar to a table in a relational database. Hive tables can store data in different formats, such as text, ORC, or Parquet, and they allow querying large datasets using HiveQL (SQL-like language).

**Types of Hive Tables:**

1. **Managed Tables:** Hive manages both the table's metadata and data. Deleting the table removes both the schema and the data.
2. **External Tables:** Hive only manages the metadata. Deleting the table does not delete the underlying data.

**Why Use Hive Tables?**

1. **Scalability:** Handle petabytes of data efficiently.
2. **Integration:** Seamlessly integrate with Hadoop's ecosystem (e.g., HDFS).
3. **SQL-Like Interface:** Easy querying using HiveQL for analysts familiar with SQL.
4. **Flexibility:** Support for structured and semi-structured data with various storage formats.

**Experiment 5**

K-Means Clustering Algorithm

**K-Means Clustering Algorithm**

K-Means is an unsupervised machine learning algorithm used for clustering. It groups data points into a predefined number of clusters, kkk, based on their similarity. The algorithm minimizes the variance within each cluster, ensuring that points in the same cluster are as similar as possible.

**Steps of K-Means Algorithm:**

1. **Initialization:**  
   Select kkk initial centroids randomly or based on some heuristic.
2. **Assignment:**  
   Assign each data point to the nearest centroid based on the Euclidean distance (or another distance metric).
3. **Update:**  
   Compute new centroids as the mean of all points assigned to each cluster.
4. **Repeat:**  
   Steps 2 and 3 are repeated until centroids stabilize (convergence) or a maximum number of iterations is reached.

**Advantages of K-Means:**

* Simple and fast for large datasets.
* Works well when clusters are spherical and evenly sized.

**Limitations:**

* Requires predefining the number of clusters (kkk).
* Sensitive to outliers and initialization.
* Assumes clusters are spherical and equally sized.

**K-Nearest Neighbors (KNN) Algorithm**

KNN is a supervised machine learning algorithm used for classification and regression. It predicts the output for a given data point based on the majority class or mean of the kkk nearest neighbors in the training data.

**Steps of KNN Algorithm:**

1. Choose the number of neighbors, kkk.
2. Calculate the distance between the input data point and all points in the training set (using Euclidean, Manhattan, or other metrics).
3. Identify the kkk closest neighbors.
4. Assign the majority class (for classification) or compute the average value (for regression).

**Advantages of KNN:**

* Simple and intuitive.
* Effective for small datasets with low noise.

**Limitations:**

* Computationally expensive for large datasets.
* Sensitive to irrelevant features and outliers.

**Key Differences: K-Means vs. KNN**

| **Aspect** | **K-Means Clustering** | **KNN Classification/Regression** |
| --- | --- | --- |
| **Type** | Unsupervised learning | Supervised learning |
| **Purpose** | Groups data into clusters | Predicts class/label for new data |
| **Output** | Clusters of similar data points | Class label or predicted value |
| **Training Phase** | No training required (iterative process) | Requires labeled training data |
| **Distance Metric** | Used to group points into clusters | Used to determine neighbors |
| **Algorithm Flow** | Iterative refinement of centroids | Instance-based (lazy learning) |

**Libraries Used**

1. **ggplot2:**
   * Visualization library for creating high-quality graphs.
   * Useful for plotting clusters and centroids.  
     *Example:* Scatter plots for visualizing clusters.
2. **factoextra:**
   * Specialized for visualizing clustering results.
   * Provides tools for determining the optimal number of clusters using methods like the Elbow method.  
     *Example:* fviz\_cluster to visualize cluster assignments.
3. **dplyr:**
   * Data manipulation library in R.
   * Useful for preprocessing and summarizing data before clustering.  
     *Example:* Filtering, grouping, and summarizing data.
4. **cluster:**
   * Provides clustering algorithms and utilities.
   * Includes methods like K-Means and others like PAM (Partitioning Around Medoids).  
     *Example:* Clustering evaluation metrics such as silhouette width.

**Experiment 6**

To compute **TF-IDF (Term Frequency-Inverse Document Frequency)** values for words from different corpora and analyze their significance in distinguishing important terms in text data.

**Key Terminologies**

**1. Corpus**

A corpus is a collection of text documents that serve as the dataset for text mining. It can consist of emails, reviews, tweets, or any other textual content.

* Example: A collection of customer reviews about a product.

**2. Term Frequency (TF)**

The **Term Frequency** of a word measures how often a term appears in a document relative to the total number of terms in that document.

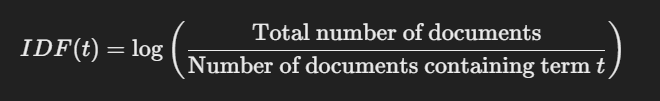
TF(t)=Number of occurrences of term t in a document / Total number of terms in the document

**Purpose:** Identifies frequently occurring terms in a document.

* **Limitation:** It does not account for how common a term might be across multiple documents.

**3. Inverse Document Frequency (IDF)**

The **Inverse Document Frequency** measures how unique or rare a term is across a collection of documents (corpus).



* **Purpose:** Gives higher weight to rare terms and lowers the weight for common terms like "the" or "is."
* **Value Range:** Higher IDF indicates rarity; lower IDF indicates commonness.

**4. TF-IDF**

The **TF-IDF** value combines TF and IDF to highlight important terms in a document that are rare across the corpus.



* **Purpose:** Identifies terms that are significant to a specific document while being uncommon across the corpus.

**5. Document-Term Matrix (DTM)**

A **Document-Term Matrix** is a matrix where rows represent documents, columns represent terms, and cell values represent the frequency of terms in the respective document.

**Libraries for TF-IDF in R**

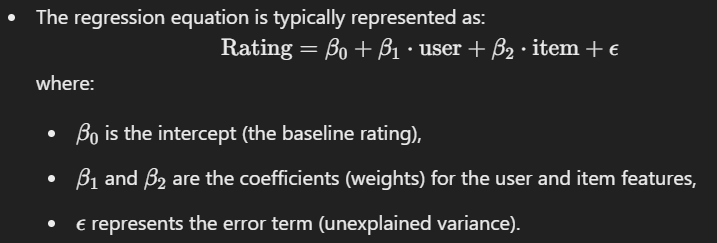
1. **tm (Text Mining):**
   * Provides tools for cleaning and processing text data.
   * Use for creating a corpus, removing stop words, and tokenization.
2. **SnowballC:**
   * Offers stemming functionality, reducing words to their root form.
3. **text2vec:**
   * Efficient for creating DTMs and calculating TF-IDF.
   * Scales well for large text corpora.
4. **tidytext:**
   * Provides a tidy framework for text mining.
   * Can compute and visualize TF-IDF values easily.

**Experiment 7**

Analytical Representation of Linear Regression using Movie Recommendation Dataset

In this experiment, we aim to build a **linear regression model** to analyze the relationship between various features (like user and movie attributes) and the ratings given by users to movies in the **MovieLens dataset**. Linear regression is one of the simplest forms of predictive modeling, and it assumes a linear relationship between the dependent and independent variables.

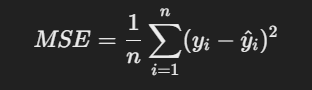
**Key Concepts and Terms Used in the Code:**

1. **MovieLens Dataset**:
   * The **MovieLens dataset** is a collection of movie ratings from users. Each row represents a user-item pair, with the rating given by a user for a specific movie. In the dataset, the columns typically include:
     + **user**: The user who rated the movie.
     + **item**: The movie that was rated.
     + **rating**: The rating given by the user to the movie.
2. **Linear Regression**:
   * **Linear Regression** is a statistical method for modeling the relationship between a dependent variable (target) and one or more independent variables (predictors). In this case, the **dependent variable** is the **rating**, and the **independent variables** are the user and movie features.
3. 

**Factors and Dummy Variables**:

* + **Factors** are categorical variables in R that represent different groups or categories. For example, the user and item variables are factors because they contain different categories of users and movies.
  + In linear regression, categorical variables need to be converted into a **numerical format** because regression models require numerical inputs. This is done by creating **dummy variables** (also called **one-hot encoding**).
    - **Dummy Variables** represent categories as binary vectors. For example, if there are 3 users, we create 3 separate columns (one for each user), where each column will have a 1 or 0 indicating whether that user is present in the row.
    - In the code, the model.matrix() function is used to convert the user and item factors into dummy variables.

1. **Model Fitting**:
   * The lm() function in R fits a linear model to the data. It estimates the coefficients (β values) for each predictor (user and item), which define how much each feature contributes to the prediction of the rating.
   * The formula rating ~ . means that we are predicting the **rating** based on all other columns (user and item variables) in the dataset.
2. **Model Summary**:
   * The summary() function provides a detailed overview of the model, including:
     + **Coefficients**: These are the values of the β parameters (intercept and weights) learned by the model. Each coefficient represents how much the corresponding feature (e.g., user or item) influences the rating prediction.
     + **Standard Error**: The standard error of the coefficient estimates, which shows how much variability exists in the estimate of each coefficient.
     + **t-value and p-value**: These values help assess the statistical significance of each predictor. A small p-value indicates that the corresponding predictor is statistically significant.
     + **R-squared**: This value measures the proportion of variance in the dependent variable (rating) that is explained by the independent variables (user and item features).
3. **Making Predictions**:
   * After fitting the model, we use the predict() function to generate predictions of ratings based on the input features.
   * In the code, predictions are made for the first 5 rows in the dataset, which represent the ratings for the first 5 user-item pairs.
4. **Model Evaluation**:
   * To evaluate the performance of the regression model, we calculate two common metrics:
     + **Mean Squared Error (MSE)**: The average of the squared differences between the actual and predicted ratings. It penalizes larger errors more heavily.



where yi​ is the actual rating, and y^i\hat{y}\_iy^​i​ is the predicted rating for the i-th observation.

* + - **Root Mean Squared Error (RMSE)**: The square root of the MSE, which provides the error in the same units as the ratings. It is easier to interpret than MSE because it returns the error in the same scale as the ratings.

**Steps in the Code Explained:**

1. **Data Loading**:
   * The MovieLense dataset is loaded using the data(MovieLense) command. This dataset is a sparse matrix of ratings.
2. **Data Conversion**:
   * The dataset is converted into a regular data frame using as(MovieLense, "data.frame"). This conversion makes it easier to manipulate and analyze the data with regression models.
3. **Preprocessing**:
   * The na.omit() function is used to remove rows with missing values (if any).
   * The user and item columns are converted into factors (categorical variables).
4. **Model Matrix Creation**:
   * The model.matrix() function is used to convert the user and item factors into dummy variables. This step is crucial for linear regression as it converts categorical data into a format that the model can use.
5. **Model Fitting**:
   * The lm() function fits a linear regression model where the dependent variable is the rating, and the independent variables are the dummy variables representing the user and item features.
6. **Prediction and Evaluation**:
   * Predictions are made for the first 5 observations using the predict() function.
   * The MSE and RMSE are calculated to evaluate the model's performance.

**Experiment 9 :**

Social Network Analysis using R (for example: Community Detection Algorithm)

**Social Network Analysis (SNA)** is a method used to analyze social structures through networks and graph theory. It uses a set of nodes (individuals or entities) connected by edges (relationships or interactions). The goal of SNA is to understand the underlying structure, identify influential nodes, and detect communities within a network.

In this experiment, we focus on **Community Detection** as a key analysis method. Community detection involves identifying groups of nodes in a network that are more densely connected to each other than to nodes outside the group. These groups are referred to as **communities**.

**Key Concepts:**

1. **Social Networks**: A social network is a representation of relationships among entities (individuals, organizations, etc.). It can be represented as a graph, where:
   * **Nodes (vertices)** represent individuals or entities.
   * **Edges (links)** represent the relationships or interactions between them.
2. **Graph Theory**: A branch of mathematics that studies the properties of graphs. A **graph** is a collection of nodes and edges, where the edges connect pairs of nodes. Social networks can be represented as undirected, weighted, or directed graphs.
3. **Community Detection**: The process of identifying groups of nodes in a graph that are more densely connected to each other than to the rest of the network. In a social network, a community could represent a group of friends, coworkers, or organizations sharing common interests.
   * **Communities** are subgroups within the network where nodes are more likely to be interconnected.
   * Community detection algorithms help in identifying these groups and understanding the social structure.

**Techniques for Community Detection:**

Several algorithms can be used to detect communities in networks, such as:

* **Louvain Method**: This is one of the most popular community detection algorithms. It is a hierarchical clustering algorithm that optimizes modularity to find communities in large networks. Modularity is a measure that quantifies the strength of division of a network into communities.
* **Girvan-Newman Algorithm**: This algorithm iteratively removes edges with the highest betweenness centrality (i.e., edges that are most important for connecting different parts of the network) until the network breaks into smaller communities.
* **Spectral Clustering**: This method uses the eigenvalues of the Laplacian matrix of the graph to partition the network into communities.

1. **Centrality Measures**: In network analysis, centrality measures help to identify the most important or influential nodes in a network. The common centrality measures include:
   * **Degree Centrality**: The number of connections a node has. Nodes with higher degree centrality are considered influential.
   * **Betweenness Centrality**: Measures how often a node acts as a bridge along the shortest path between two other nodes.
   * **Closeness Centrality**: Measures the distance from a node to all other nodes in the network. Nodes with high closeness centrality are considered to be well-positioned in terms of information flow.
2. **Cliques**: A clique is a subset of nodes in a graph such that every two nodes are adjacent to each other (i.e., they form a complete subgraph). Identifying cliques can help detect tightly-knit groups within a network.
3. **Network Visualization**: Visualization tools like the igraph package in R are often used to visualize networks and communities. Different nodes can be sized or colored based on centrality measures or community assignments to provide better insights.

**Loading Libraries**:

* igraph: The main library used for social network analysis in R. It provides tools for creating, manipulating, and analyzing graphs.
* igraphdata: This package provides various datasets, including the well-known **Karate Club** network dataset used in this experiment.