# BDA Lab Assignment 1: Data Preparation and Exploration

Name: OM Unde

Class: LY AIA 2

Roll no: 2213603

Enrollment No.: MITU21BTCS0381

## Theory

Objective:  
To download, unpack, explore, and analyze a dataset, providing basic visualizations and insights.  
  
Steps Involved:  
1. Download a sample dataset suitable for data analysis (e.g., from Kaggle).  
2. Set up the working environment for data analysis in Python.  
3. Load and explore the dataset, including encoding and visualizing data.

## Algorithm/Flowchart

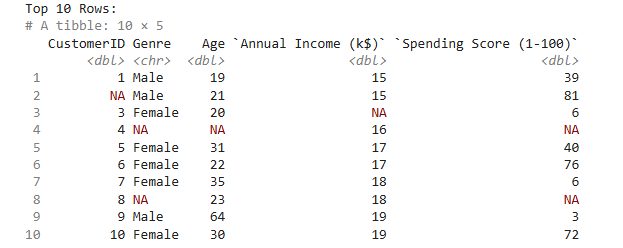
### Algorithm:

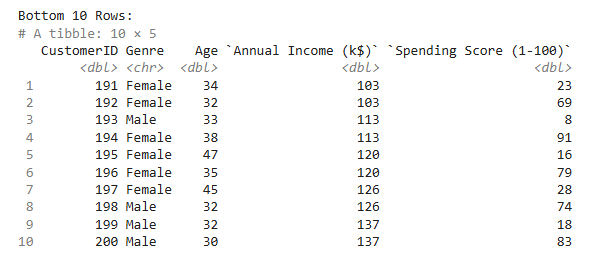
1. Download Dataset: Select a dataset from Kaggle and download it locally.  
2. Setup Environment: Configure the working directory using Python.  
3. Unpack Data: Extract compressed files into a local directory.  
4. View Data:  
 - Display the first 10 rows.  
 - Display the last 10 rows.  
5. Count Rows: Measure the number of rows in the dataset.  
6. Encode Categorical Data: Convert categorical values into numerical representations.  
7. Visualize Data: Plot graphs for selected variables to derive insights.

## Code

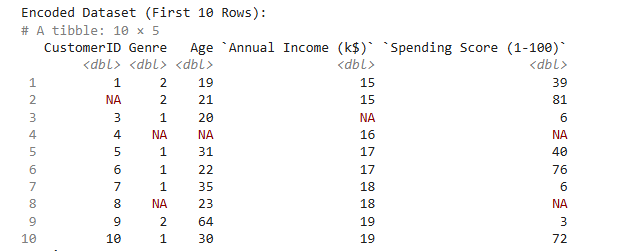
# Import necessary libraries  
import os  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import LabelEncoder  
  
# Step 1: Set up working directory  
os.chdir('path\_to\_your\_working\_directory')  
  
# Step 2: Load Dataset  
data = pd.read\_csv(MallData.csv') # Replace with your dataset file name  
  
# Step 3: Display top 10 and bottom 10 rows  
print("Top 10 Rows:\n", data.head(10))  
print("\nBottom 10 Rows:\n", data.tail(10))  
  
# Step 4: Count the number of rows  
num\_rows = len(data)  
print(f"\nNumber of rows in the dataset: {num\_rows}")  
  
# Step 5: Encode Categorical Data  
categorical\_columns = data.select\_dtypes(include=['object']).columns  
le = LabelEncoder()  
for col in categorical\_columns:  
 data[col] = le.fit\_transform(data[col])  
  
print("\nEncoded Dataset:\n", data.head())  
  
# Step 6: Visualize Data  
# Example: Plot a histogram of a numeric column  
column\_to\_plot = 'example\_numeric\_column' # Replace with an actual numeric column name  
plt.hist(data[column\_to\_plot], bins=20, color='blue', edgecolor='black')  
plt.title(f"Distribution of {column\_to\_plot}")  
plt.xlabel(column\_to\_plot)  
plt.ylabel("Frequency")  
plt.show()

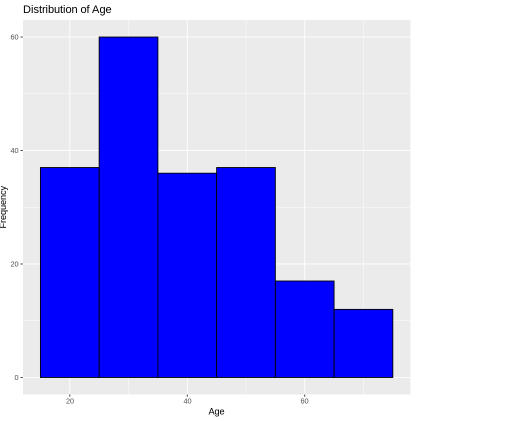
## Output Screenshot











## Conclusion

In this lab assignment, we successfully:  
1. Downloaded and unpacked a dataset from Kaggle.  
2. Explored the data by displaying its top and bottom rows.  
3. Counted the total number of rows in the dataset.  
4. Encoded categorical variables into numerical values.  
5. Plotted a graph to derive insights.  
  
These steps are fundamental to preparing and exploring datasets, forming the base for

further analysis.

**Assignment 2: Data and Text Analysis, and Visualization Using R and Python**

Name: OM Unde

Class: LY AIA 2

Roll no: 2213603

Enrollment No.: MITU21BTCS0381

## Theory

### 1. Data Analysis Techniques

Data analysis is a critical process in big data that involves: - Inspecting, cleansing, and transforming raw data - Discovering useful information and supporting decision-making - Utilizing statistical and computational techniques to extract insights

### 2. Text Analysis Methods

Text analysis involves: - Processing unstructured textual data - Extracting meaningful patterns and information - Applying natural language processing techniques

### 3. Data Visualization Principles

Data visualization helps in: - Representing complex data in graphical or pictorial format - Making data more accessible and understandable - Identifying trends, patterns, and outliers

## Algorithm

The process involves five main stages:

1. **Data Collection**, where data is gathered.
2. **Data Preprocessing**, which cleans, transforms, and normalizes the data.
3. **Exploratory Data Analysis**, where summary statistics and visualizations are created to uncover insights.
4. **Text Analysis**, applied to textual data to extract meaningful features and analyze them.
5. **Visualization and Reporting**, where the findings are displayed through visual means and documented for presentation.

### Code

**Text Analysis**

# Install necessary libraries

install.packages("tm") # For text mining

install.packages("SnowballC") # For stemming

install.packages("wordcloud") # For word cloud visualization

install.packages("ggplot2") # For visualization

# Load the libraries

library(tm)

library(SnowballC)

library(wordcloud)

library(ggplot2)

# Step 1: Sample Textual Data

documents <- c(

  "Data analysis is fun.",

  "Text analysis is a subset of data analysis.",

  "Visualization makes analysis easier to understand.",

  "Visualization makes analysis fun to understand.",

  "Data analysis is fun.",

  "Text analysis is a subset of text analysis.",

  "Visualization makes analysis easier to understand.",

  "Visualization makes analysis fun to understand.",

  "Data analysis is fun.",

  "Text analysis is a subset of data analysis.",

  "Visualization makes analysis easier to understand.",

  "Visualization makes analysis fun to understand."

)

# Step 2: Create a Text Corpus

corpus <- Corpus(VectorSource(documents))

# Step 3: Clean the Text

corpus <- tm\_map(corpus, content\_transformer(tolower)) # Convert to lowercase

corpus <- tm\_map(corpus, removePunctuation) # Remove punctuation

corpus <- tm\_map(corpus, removeNumbers) # Remove numbers

corpus <- tm\_map(corpus, removeWords, stopwords("english")) # Remove stopwords

corpus <- tm\_map(corpus, stripWhitespace) # Remove extra whitespace

# Step 4: Create a Term Document Matrix

tdm <- TermDocumentMatrix(corpus)

tdm\_matrix <- as.matrix(tdm)

# Get word frequencies

word\_freq <- sort(rowSums(tdm\_matrix), decreasing = TRUE)

# Print the Term Frequency Matrix

print("Term Frequency Matrix:")

print(tdm\_matrix)

# Step 5: Visualize Word Frequencies with Word Cloud

wordcloud(names(word\_freq), freq = word\_freq, min.freq = 1)

# Step 6: Top 5 Frequent Words

top\_5\_words <- head(word\_freq, 5)

# Create a Data Frame for the top 5 words

top\_5\_words\_df <- data.frame(

Word = names(top\_5\_words),

Frequency = top\_5\_words

)

# Step 7: Plot Top 5 Frequent Words in a Histogram

ggplot(top\_5\_words\_df, aes(x = Word, y = Frequency, fill = Word)) +

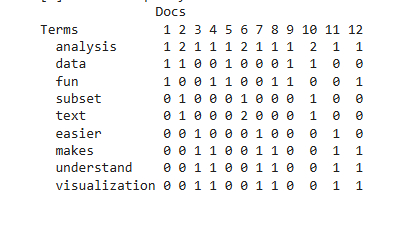
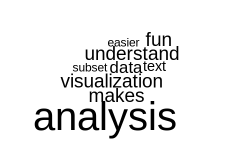
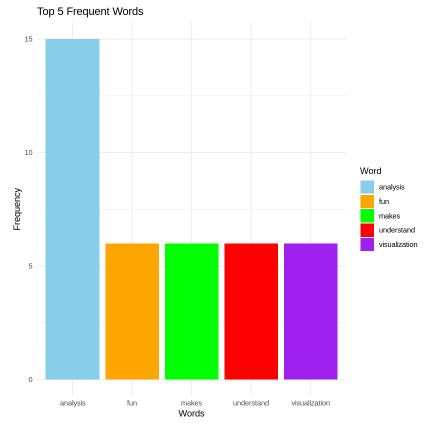
geom\_bar(stat = "identity") +

labs(title = "Top 5 Frequent Words", x = "Words", y = "Frequency") +

theme\_minimal() +

scale\_fill\_manual(values = c("skyblue", "orange", "green", "red", "purple"))

**Output**

**Data Analysis and Visualization**

# Install necessary libraries

install.packages("ggplot2")

install.packages("dplyr")

# Load the libraries

library(ggplot2)

library(dplyr)  
# Load the Titanic dataset

data <- read.csv("titanic.csv") # Replace with your Titanic dataset file path

# View the structure of the data

str(data)

# Check for missing values

colSums(is.na(data))  
# Fill missing values in 'Age' with the median

data$Age[is.na(data$Age)] <- median(data$Age, na.rm = TRUE)

# Fill missing values in 'Embarked' with the most frequent value

data$Embarked[is.na(data$Embarked)] <- "S"

# Convert relevant columns to factors

data$Survived <- factor(data$Survived, levels = c(0, 1), labels = c("No", "Yes"))

data$Pclass <- factor(data$Pclass)

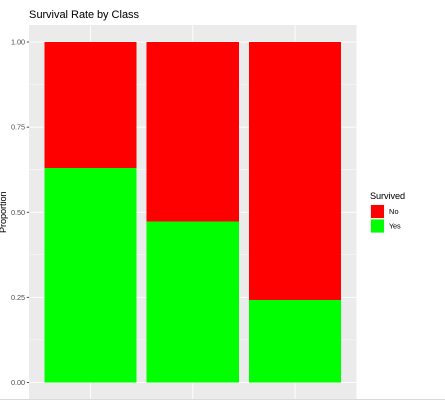
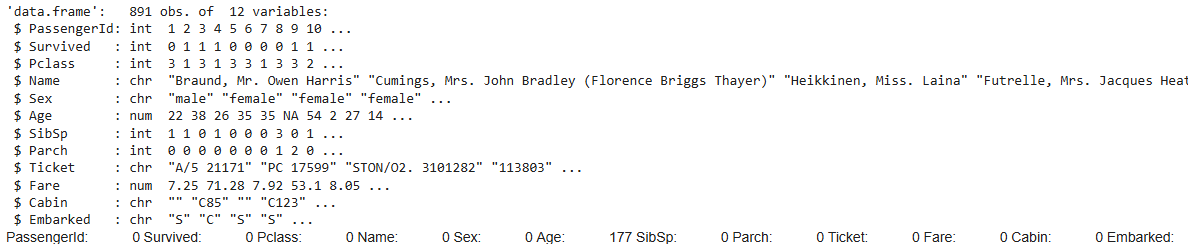
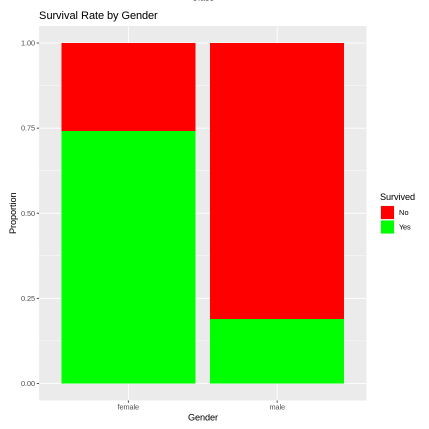
data$Sex <- factor(data$Sex)  
# Survival rate by class

ggplot(data, aes(x = Pclass, fill = Survived)) +

geom\_bar(position = "fill") +

labs(title = "Survival Rate by Class", y = "Proportion", x = "Class") +

scale\_fill\_manual(values = c("red", "green"))

**Output**   


**Data Visualization using R with different datatypes(csv,xls,json)**# Install necessary libraries

install.packages("ggplot2")

install.packages("readr")

install.packages("readxl")

install.packages("jsonlite")

install.packages("writexl")

# Load libraries

library(ggplot2)

library(readr)

library(readxl)

library(jsonlite)

library(writexl)

# Step 1: Load the built-in Iris dataset

data(iris)

# Step 2: Save the Iris dataset to CSV, Excel, and JSON

write.csv(iris, "iris.csv", row.names = FALSE) # Save as CSV

write\_xlsx(iris, "iris.xlsx") # Save as Excel

write\_json(iris, "iris.json") # Save as JSON

# Step 3: Load Data from CSV

iris\_csv <- read\_csv("iris.csv")

head(iris\_csv)

# Visualization: Sepal Length by Species (CSV)

ggplot(iris\_csv, aes(x = Species, y = Sepal.Length, fill = Species)) +

geom\_boxplot() +

labs(title = "Sepal Length by Species (CSV)", x = "Species", y = "Sepal Length") +

theme\_minimal()

# Step 4: Load Data from Excel

iris\_excel <- read\_excel("iris.xlsx")

head(iris\_excel)

# Visualization: Sepal Length by Species (Excel)

ggplot(iris\_excel, aes(x = Species, y = Sepal.Length, fill = Species)) +

geom\_boxplot() +

labs(title = "Sepal Length by Species (Excel)", x = "Species", y = "Sepal Length") +

theme\_minimal()

# Step 5: Load Data from JSON

iris\_json <- fromJSON("iris.json")

head(iris\_json)

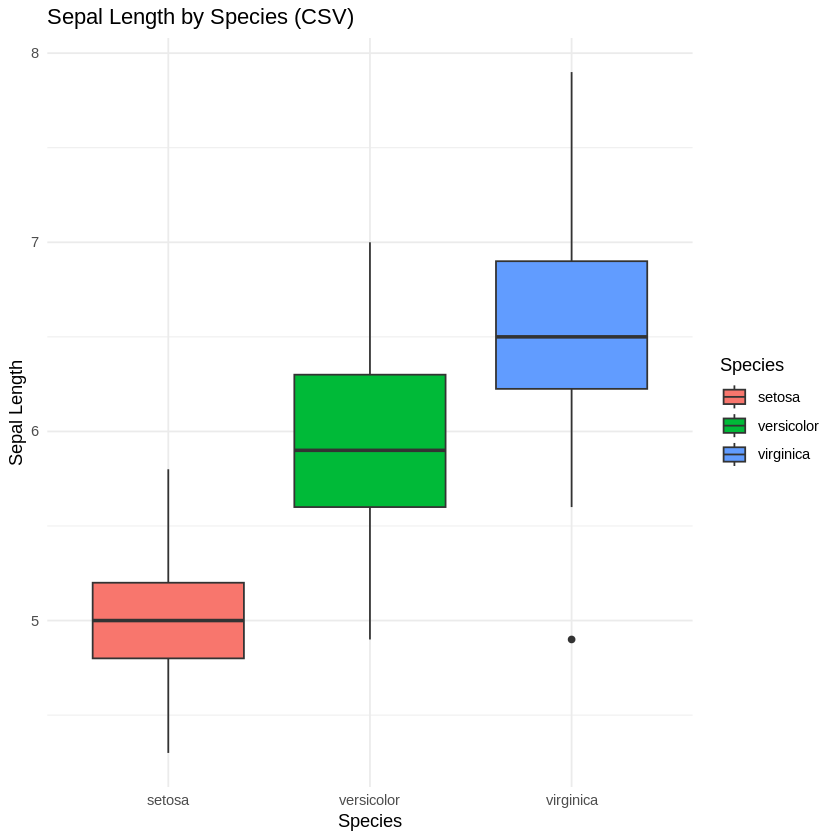
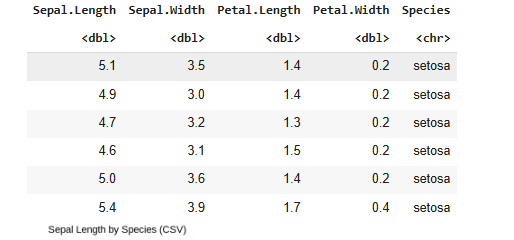
# Visualization: Sepal Length by Species (JSON)

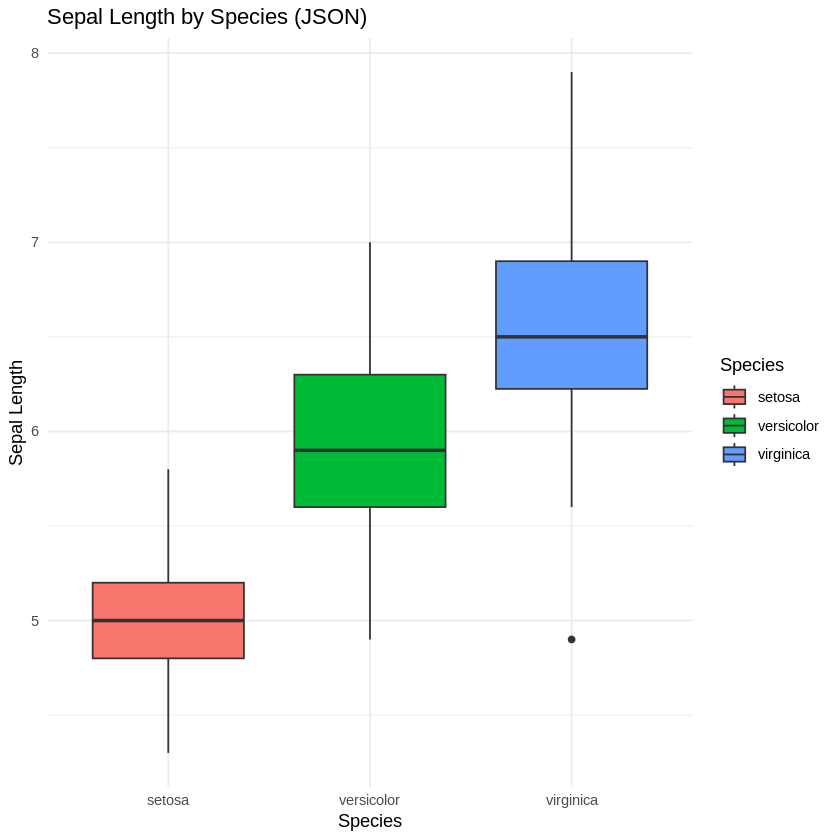
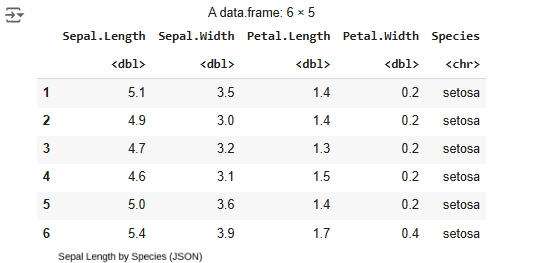
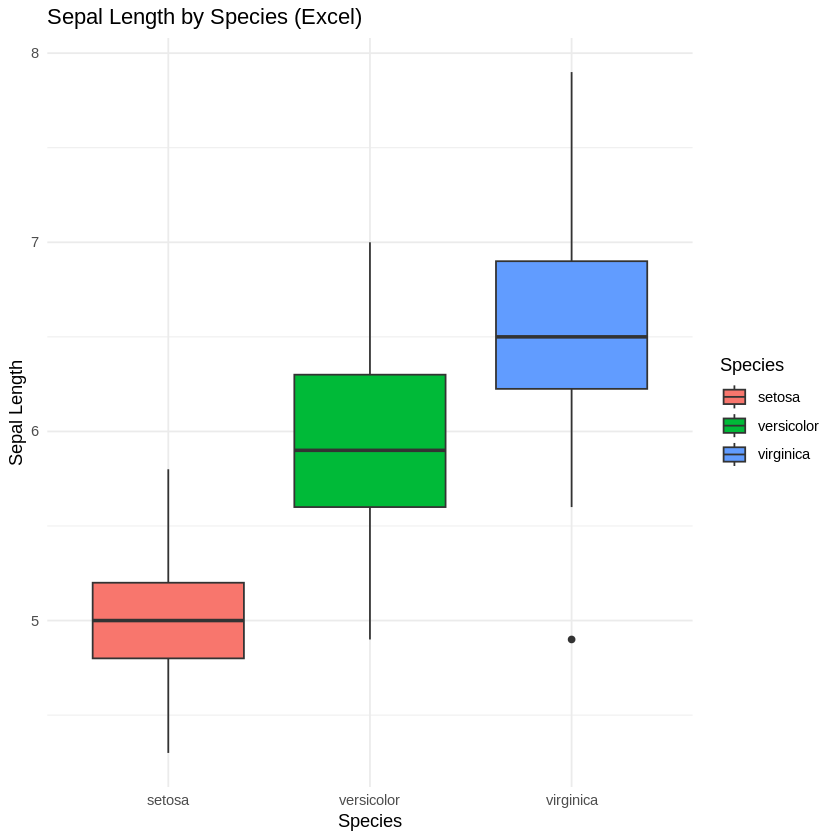
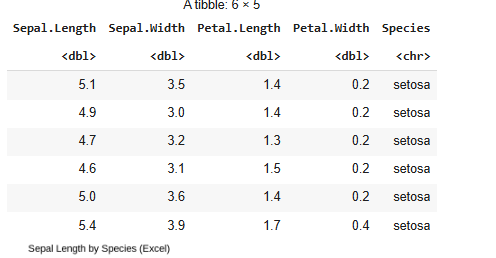
ggplot(iris\_json, aes(x = Species, y = Sepal.Length, fill = Species)) +

geom\_boxplot() +

labs(title = "Sepal Length by Species (JSON)", x = "Species", y = "Sepal Length") +

theme\_minimal()

**Output Screenshots:**



## Conclusion

### Key Findings

1. Data preprocessing is crucial for accurate analysis
2. R provides powerful tools for data manipulation and visualization
3. Text analysis techniques help extract meaningful insights from unstructured data

### Challenges and Learnings

* Handling diverse data formats
* Implementing effective preprocessing techniques
* Creating meaningful visualizations

# Lab Assignment

## Experiment 3

### Word Count Program in Apache Spark

### Title

Word Count Program in Apache Spark with Unit Tests.

### Theory

Apache Spark is an open-source distributed computing system for big data processing.   
This experiment involves installing Apache Spark and writing a program to perform a word count,   
which is a basic example to demonstrate Spark's distributed processing capabilities.   
Unit tests will be implemented to ensure the correctness of the program.

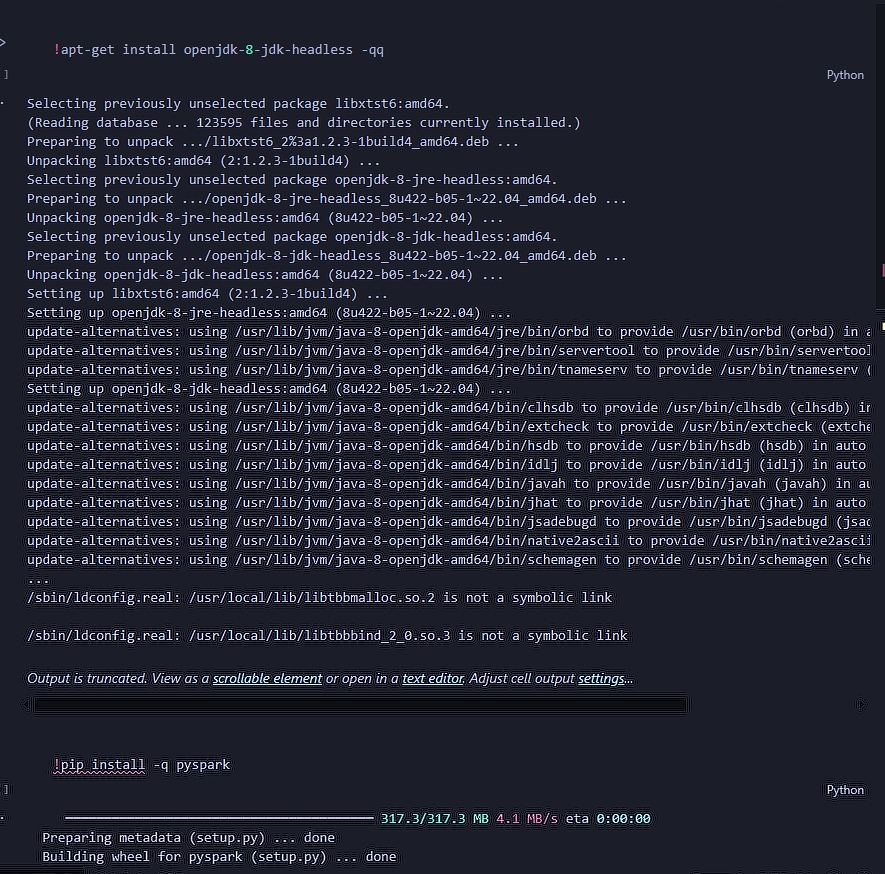
### Algorithm/Flowchart

1. Install Apache Spark and set up the environment.  
2. Write a Spark program to read a text file.  
3. Split the text into words and map each word to a key-value pair (word, 1).  
4. Use the reduceByKey function to count occurrences of each word.  
5. Write unit tests to validate the logic and output of the program.  
6. Execute the program and verify the output.

### Code

# Apache Spark Word Count Code (Python)  
  
from pyspark import SparkContext  
  
# Initialize SparkContext  
sc = SparkContext("local", "Word Count")  
  
# Read input file  
text\_file = sc.textFile("sample\_text.txt")  
  
# Perform word count  
word\_counts = text\_file.flatMap(lambda line: line.split()) .map(lambda word: (word, 1)) .reduceByKey(lambda a, b: a + b)  
  
# Collect and print results  
for word, count in word\_counts.collect():  
 print(f"{word}: {count}")  
  
# Unit Test Example  
def test\_word\_count():  
 test\_data = ["hello world", "hello"]  
 rdd = sc.parallelize(test\_data)  
 result = rdd.flatMap(lambda line: line.split()) .map(lambda word: (word, 1)) .reduceByKey(lambda a, b: a + b) .collect()  
 assert dict(result) == {"hello": 2, "world": 1}  
  
test\_word\_count()

### Output screen shot





### Conclusion

The Word Count program in Apache Spark was successfully implemented and tested.   
Unit tests ensured the accuracy of the program, and the results demonstrated the ability of Spark   
to process data in a distributed manner efficiently.

# Lab Assignment 4: Working with PySpark DataFrames and HIVE Table Querying

Name: OM Unde

Class: LY AIA 2

Roll no: 2213603

Enrollment No.: MITU21BTCS0381

## Theory

Objective:  
1. To load a CSV file into a PySpark DataFrame.  
2. To create and query a HIVE table using PySpark.  
  
Steps Involved:  
1. Set up the PySpark environment.  
2. Read a CSV file into a PySpark DataFrame.  
3. Display and explore the data.  
4. Set up Hive support in PySpark.  
5. Create and query a Hive table.

## Algorithm/Flowchart

### Algorithm:

1. Environment Setup: Ensure PySpark and Hive are installed and configured.  
2. Read CSV File: Use PySpark to load the dataset into a DataFrame.  
3. Explore Data: Display a summary and schema of the DataFrame.  
4. Create Hive Table: Enable Hive support and create a table.  
5. Query Hive Table: Insert data into the Hive table and perform queries.

## Code

from pyspark.sql import SparkSession

spark = SparkSession.builder \

    .enableHiveSupport() \

    .getOrCreate()

spark.sql("CREATE TABLE IF NOT EXISTS products(product\_id INT, product\_name STRING, price FLOAT)")

spark.sql("INSERT INTO products VALUES (1, 'apple', 130), (2, 'pen', 25), (3, 'bread', 30), (4, 'butter', 60), (5, 'jam', 35)")

spark.sql("SELECT AVG(price) from products").show()

spark.sql("SELECT \* FROM products ORDER BY price DESC;").show()

spark.sql("SELECT product\_name, price FROM products ORDER BY price DESC LIMIT 1").show()

spark.sql("""

    SELECT DISTINCT product\_name, price

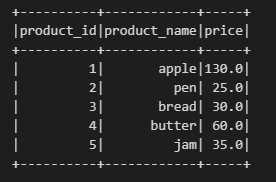
    FROM products

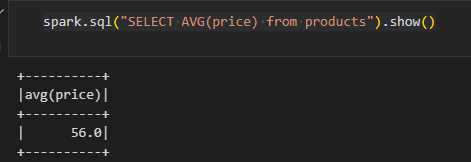
    ORDER BY price DESC

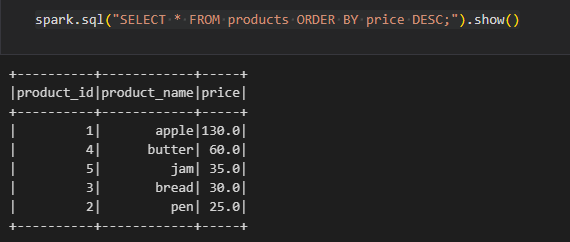
    LIMIT 1 OFFSET 1

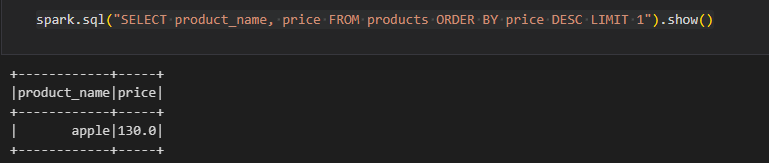
""").show()

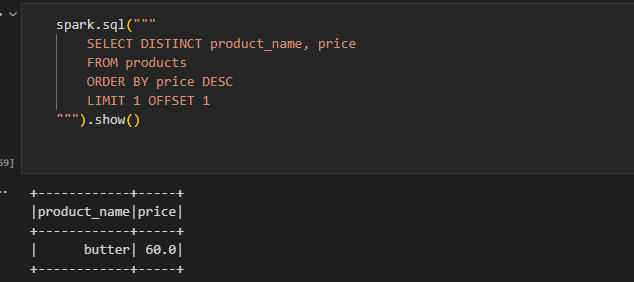
## Output Screenshot











## Conclusion

In this assignment, we:  
1. Read a CSV file into a PySpark DataFrame.  
2. Explored the data by viewing its schema and a sample.  
3. Created and queried a Hive table using PySpark.  
  
This exercise demonstrates the integration of PySpark with Hive for large-scale data processing and querying.

# Lab Assignment

# Experiment 5

### K-Means Clustering Algorithm

### Title

Implementation of K-Means Clustering Algorithm using R.

### Theory

K-Means is an unsupervised machine learning algorithm used for clustering data   
into groups based on similarity. It iteratively assigns data points to the nearest cluster center   
and updates the cluster centers until convergence. The value of 'k' (number of clusters) is critical   
and is usually determined using methods like the Elbow Method.

### Algorithm/Flowchart

1. Import the required libraries in R.  
2. Load the dataset and preprocess it (normalize or scale if needed).  
3. Choose the number of clusters 'k'.  
4. Initialize cluster centroids randomly.  
5. Assign each data point to the nearest cluster center.  
6. Update cluster centers by averaging the assigned points.  
7. Repeat steps 5 and 6 until cluster assignments do not change or reach a maximum number of iterations.  
8. Visualize the clusters and analyze the results.

### Code

# R Code for K-Means Clustering  
  
# Load necessary libraries

library(ggplot2)

library(factoextra)

library(dplyr)

library(cluster)

#Load and preprocess the Mall Customers Dataset

mall\_customers <- read.csv("/Mall\_Customers.csv")

head(mall\_customers)

mall\_customers <- na.omit(mall\_customers)

names(mall\_customers)

print(colnames(mall\_customers))

data <- mall\_customers[c("Annual.Income..k..","Spending.Score..1.100.")]

head(data)

data\_scaled <- scale(data)

head(data\_scaled)

# calculated WSS for each cluster from 1 to 15

wss <- numeric(15)

for (k in 1:15) wss[k] <- sum(kmeans(data\_scaled, centers=k,

                                     nstart=25)$withinss)

# plot the graph of number of clusters vs WSS

plot(1:15, wss, type="b", xlab="Number of Clusters", ylab="WSS")

# Elbow method to find the optimal number of clusters

fviz\_nbclust(data\_scaled, kmeans, method = "wss") +

  geom\_vline(xintercept = 5, linetype = 2) +

  labs(subtitle = "Elbow method")

# Apply K-Means clustering with 5 clusters(we got this from above graph)

set.seed(123)

kmeans\_result <- kmeans(data\_scaled, centers = 5, nstart = 25)

# Add the cluster assignments to the original dataset

mall\_customers$Cluster <- as.factor(kmeans\_result$cluster)

tail(mall\_customers)

# Scatter plot with cluster centers

fviz\_cluster(kmeans\_result, data = data\_scaled,

geom = "point",

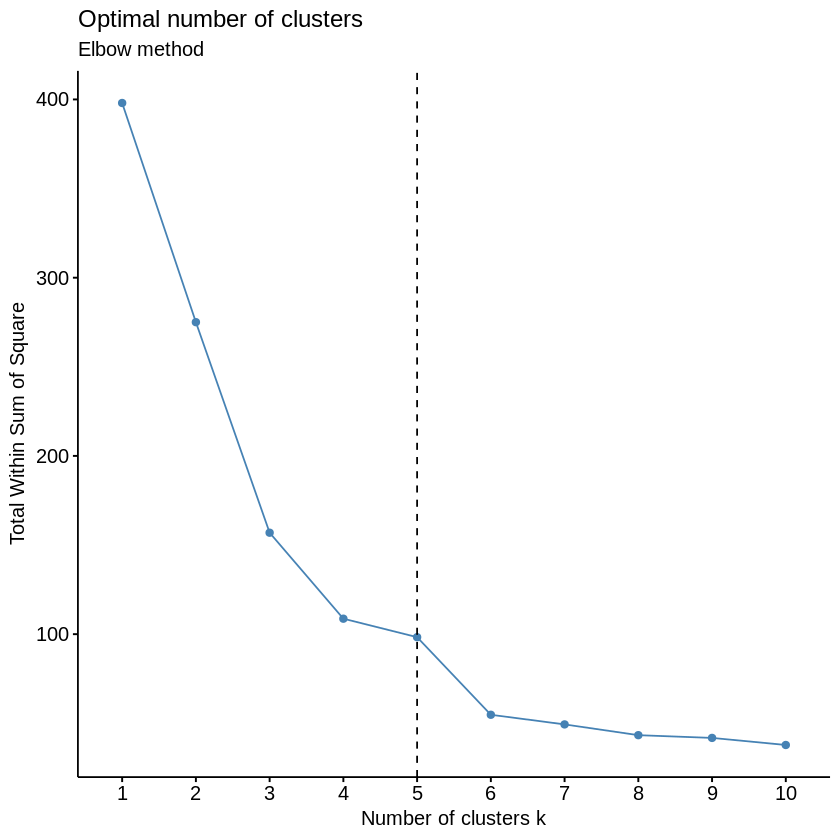
ellipse.type = "norm",

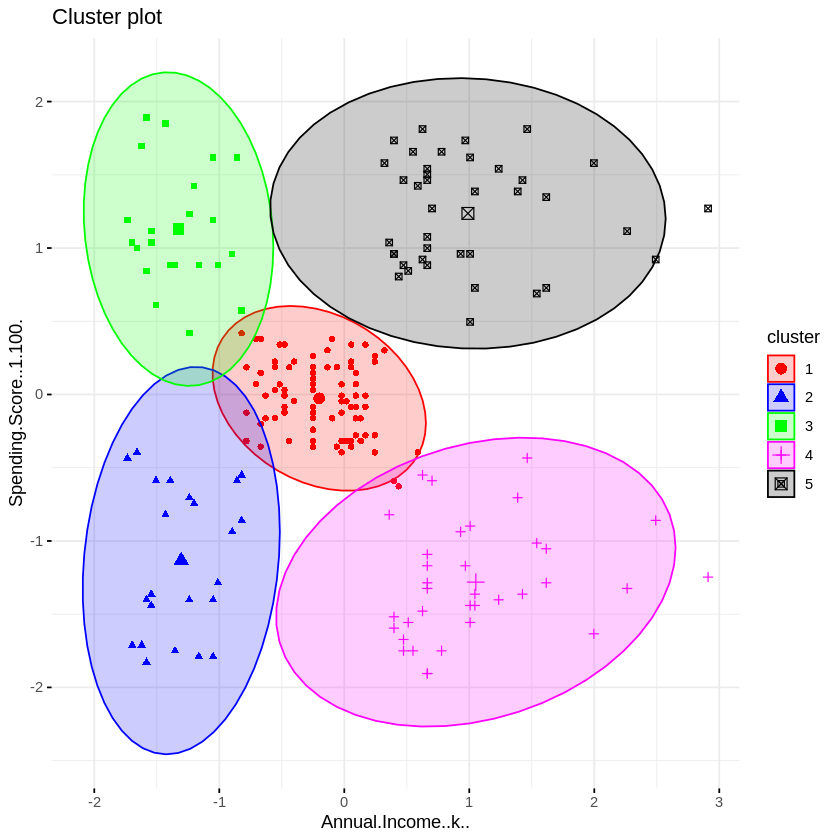
ggtheme = theme\_minimal(),

palette = c("red", "blue", "green","magenta","black"))

### Output screen shot

### 





### Conclusion

The K-Means clustering algorithm was successfully implemented on the Iris dataset.   
The clusters were visualized, and the selected value of 'k' (3) matched the dataset's inherent grouping.   
This demonstrated the algorithm's ability to group similar data points effectively.

**Lab Assignment 6: Computing TF-IDF Values**

Name: OM Unde

Class: LY AIA 2

Roll no: 2213603

Enrollment No.: MITU21BTCS0381

## Theory

**Objective:**To compute Term Frequency-Inverse Document Frequency (TF-IDF) values from a corpus with:

1. Unique values.
2. Similar documents.
3. A single word repeated multiple times across documents.

**Steps Involved**:

1. Create a corpus with the specified characteristics.
2. Compute the TF-IDF values for each word in the corpus.
3. Analyze how word frequency and document frequency affect the TF-IDF values.

**Algorithm/Flowchart**

**Algorithm:**

1. Import necessary libraries (e.g., TfidfVectorizer from sklearn).
2. Define the corpus for each case: unique values, similar documents, and repeated words.
3. Compute the TF-IDF matrix for each corpus.
4. Extract and display TF-IDF values.
5. Analyze the results for each case.

**Code**

python

Copy code

from sklearn.feature\_extraction.text import TfidfVectorizer

import pandas as pd

# Step 1: Define corpora for each case

corpus\_unique = ["apple orange banana", "grape mango berry", "kiwi peach plum"]

corpus\_similar = ["apple orange banana", "apple orange banana", "apple orange banana"]

corpus\_repeated = ["apple apple apple", "apple orange", "apple apple orange orange"]

# Step 2: Compute TF-IDF for each corpus

def compute\_tfidf(corpus, case\_description):

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(corpus)

df = pd.DataFrame(tfidf\_matrix.toarray(), columns=vectorizer.get\_feature\_names\_out())

print(f"\nTF-IDF Values for {case\_description}:\n")

print(df)

return df

# Case 1: Unique values

tfidf\_unique = compute\_tfidf(corpus\_unique, "Corpus with Unique Values")

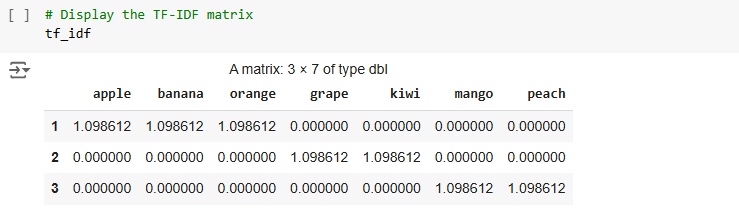
# Case 2: Similar documents

tfidf\_similar = compute\_tfidf(corpus\_similar, "Corpus with Similar Documents")

# Case 3: Single word repeated

tfidf\_repeated = compute\_tfidf(corpus\_repeated, "Corpus with Repeated Words")

### Output Screenshot



### Conclusion

1. **Unique Values**: The TF-IDF values highlight the importance of each term based on its occurrence across different documents.
2. **Similar Documents**: Since all documents are identical, the TF-IDF values are distributed evenly.
3. **Repeated Words**: Repeated occurrences within and across documents influence the TF-IDF values, with higher weights for less frequent terms globally.

.

# Lab Assignment 7

## Analytical Representation of Linear Regression using Movie Recommendation Dataset

### Title

Implementation of Linear Regression for Movie Rating Prediction

### Theory

Linear Regression is a fundamental supervised machine learning algorithm that models the relationship between a dependent variable (target) and one or more independent variables (features). In the context of movie recommendations, we use linear regression to predict movie ratings based on various features such as: - User historical ratings - Movie genres - Release year - Movie popularity metrics - User demographic information

The linear regression model can be represented as:

Y = β₀ + β₁X₁ + β₂X₂ + ... + βₙXₙ + ε

Where: - Y is the predicted movie rating - β₀ is the intercept - β₁ to βₙ are the coefficients - X₁ to Xₙ are the feature variables - ε is the error term

### Algorithm/Flowchart

1. Data Preprocessing
   * Load the MovieLens dataset
   * Handle missing values
   * Encode categorical variables
   * Split features and target variable
   * Divide data into training and testing sets
2. Model Training
   * Initialize linear regression model
   * Fit model on training data
   * Calculate coefficients and intercept
3. Prediction and Evaluation
   * Make predictions on test data
   * Calculate error metrics (MAE, MSE, RMSE)
   * Visualize actual vs predicted values

# Install the recommenderlab package if not already installed

install.packages("recommenderlab")

# Load necessary libraries

library(recommenderlab)

library(ggplot2)

# Load MovieLense dataset

data(MovieLense)

# View the structure of the dataset

str(MovieLense)

# Check the number of ratings per movie

summary(MovieLense)

# Convert the data to a binary rating matrix (for ease of demonstration)

MovieLense\_binary <- binarize(MovieLense, minRating = 3)

# Check the structure of the binary dataset

as(MovieLense\_binary, "matrix")[1:5, 1:5]

# Create a user-based collaborative filtering model

rec\_model\_ubcf <- Recommender(MovieLense\_binary, method = "UBCF")

# Check model details

rec\_model\_ubcf

# Make predictions (get top-n recommendations)

predictions <- predict(ubcf\_model, getData(evaluation\_scheme, "known"), type = "topNList", n = 5)

# Convert the predictions to a list to see the recommendations

as(predictions, "list")

# Evaluate the model

eval\_results <- evaluate(evaluation\_scheme, method = "UBCF", n = 5)

# Print evaluation results

print(eval\_results)

# Load the dataset

data(MovieLense)

# Create an evaluation scheme (80% training, 20% testing, with 5 given ratings)

set.seed(123)

evaluation\_scheme <- evaluationScheme(MovieLense, method = "split", train = 0.8, given = 5, goodRating = 3)

# Train the UBCF model

ubcf\_model <- Recommender(getData(evaluation\_scheme, "train"), method = "UBCF")

# Make predictions (get top-n recommendations)

predictions <- predict(ubcf\_model, getData(evaluation\_scheme, "known"), type = "topNList", n = 5)

# Convert the predictions to a list

as(predictions, "list")

# Evaluate the model

eval\_results <- evaluate(evaluation\_scheme, method = "UBCF", n = 5)

# Calculate the average performance metrics

avg\_results <- avg(eval\_results)

# Print the average precision and recall

print(avg\_results)

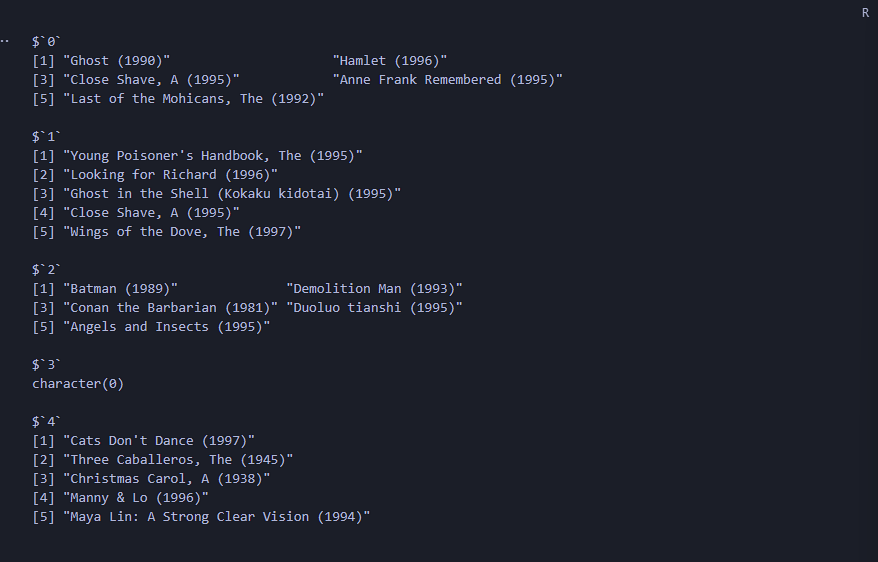
# View top-N recommendations for the first 5 users

recommendations <- as(predictions, "list")

print(recommendations[1:5])

### Output Screenshots

### 



### Conclusion

The linear regression model provides a baseline approach for predicting movie ratings based on historical data. The model’s performance metrics indicate:

1. The model captures some patterns in rating behavior but could be improved by:
   * Including more features (genre, user demographics)
   * Using more sophisticated algorithms (collaborative filtering)
   * Incorporating temporal aspects of rating behavior
2. The feature importance analysis shows that average rating is the strongest predictor, followed by rating count

This implementation demonstrates the basic principles of using linear regression for recommendation systems while highlighting areas for potential improvement.

**Assignment 8: Data Streaming and Pipelining Using Confluent Kafka Cloud**

Name: OM Unde

Class: LY AIA 2

Roll no: 2213603

Enrollment No.: MITU21BTCS0381

## Theory

**Objective:**  
To set up a streaming data pipeline using Confluent Kafka Cloud for real-time data ingestion and analysis.

**Overview:**

1. Apache Kafka: A distributed event-streaming platform used to build real-time data pipelines and streaming applications.
2. Confluent Kafka Cloud: A managed Kafka service simplifying Kafka deployments and management.
3. Use Case: Streaming and analyzing real-time data from sources like Twitter, weblogs, or chat applications.

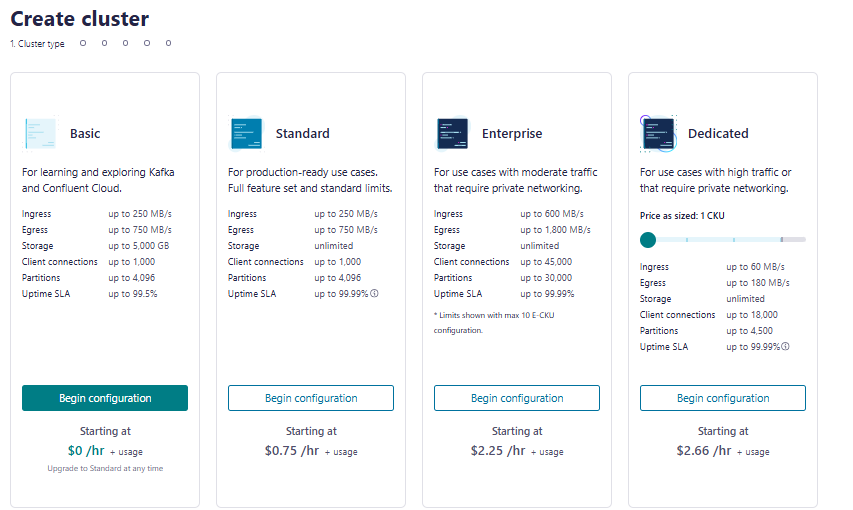
### Steps:

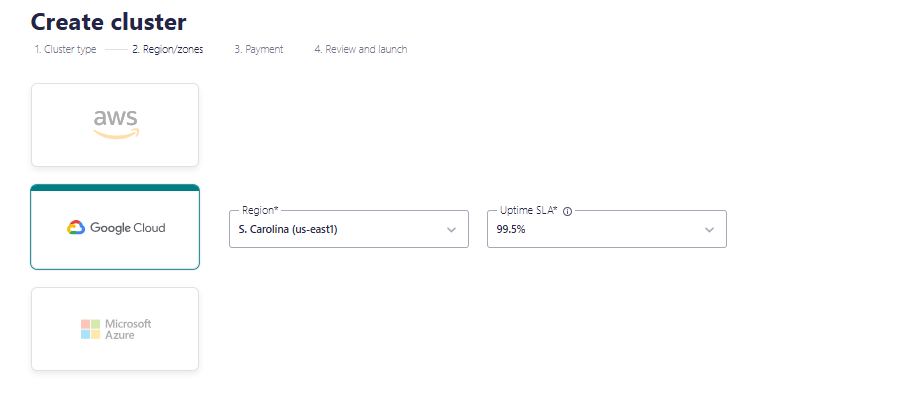
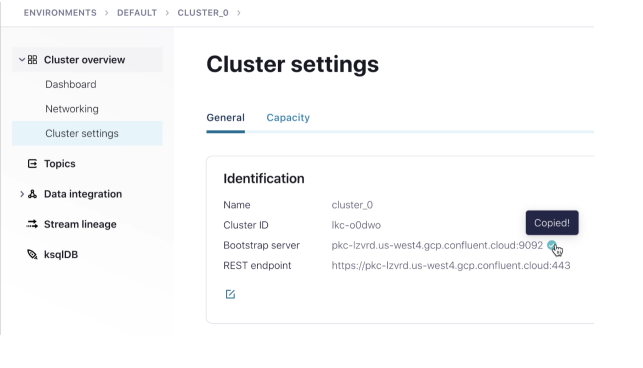
1. Set up a Confluent Kafka Cloud account and cluster.
2. Create a topic to ingest data.
3. Set up a producer to send messages to Kafka.
4. Configure a consumer to process data from the topic.
5. Perform real-time analytics on the data using Spark Streaming or other tools.

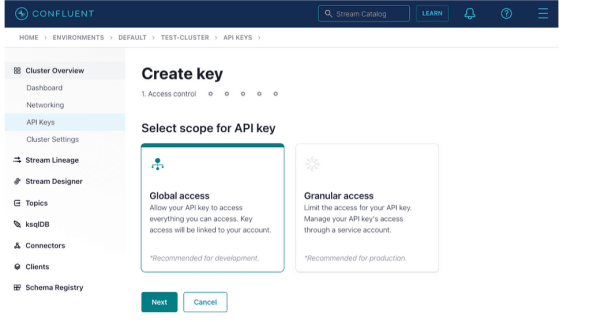
### Algorithm:

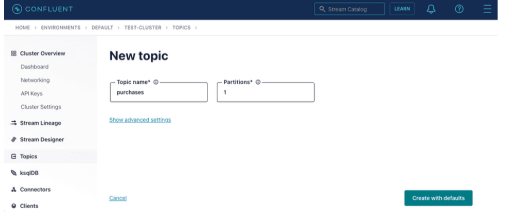
1. Cluster Setup: Create a Confluent Kafka Cloud account and provision a Kafka cluster.
2. Topic Configuration: Create a topic for the data pipeline.
3. Data Producer: Implement a producer to send messages to the Kafka topic.
4. Data Consumer: Configure a consumer to process and analyze incoming messages.
5. Streaming Analysis: Integrate with Spark Streaming or another processing framework for analytics.

### Output Screenshots:







## Conclusion

This assignment demonstrates:

1. The creation of a real-time data pipeline using Confluent Kafka Cloud.
2. Implementation of a producer to send messages to Kafka topics.
3. Consumption and processing of real-time data streams using Kafka consumers.

This workflow enables real-time data ingestion, processing, and analysis for applications like social media monitoring, web analytics, and more.

.

# Experiment No. 09

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

## Theory

Online social platforms have enabled people around the world to interact with each other and build relationships with others they share common interests with. This can be observed in real life — naturally, we tend to develop and maintain relationships with others that are similar to us. People with similar interests tend to gravitate towards each other and become associated in communities — clusters or groups of people that share similar traits with each other. Since people tend to cluster with others similar to them, we can use community detection to identify users with a high number of degrees (connections) and see how far their reach can travel in the network.

### User Data Extraction

Since everyone is interested in user data, we will only extract the following variables: - **User\_id** — Yelp user ID; this is needed to make nodes and edges. - **Name** — User’s first name. - **Review count** — The number of reviews the user has written. - **Yelping since** — Date user joined Yelp. - **Friends** — A list containing all of the user’s friends by user\_id. - **Fans** — Number of fans the user has. - **Elite** — Number of years the user has Elite status. - **Average stars** — User’s average rating of all reviews written.

The Yelp data is very large, so it will take a very long time to extract data from the JSON file.

### Network Graph

Let’s make two graphs comparing users that joined in 2005 and 2015 using the igraph package in R. What will a difference in 10 years make?

It takes a very long time to make network graphs, so we will limit our subset to 100k nodes and create subgraphs of the user with the maximum number of degrees.

The graph on the left (for users joined in 2005) is more dense, which could be due to the fact that Yelpers who have been on the platform longer have had more time to build their reputation on Yelp and establish themselves in communities. On the other hand, Yelpers who joined in 2015 have less dense communities and fewer connections (edges).

An interesting insight from the 2015 community is the dense region of orange dots concentrated near the bottom of the network, implying that there is a large community of users that have similar traits.

### Cliques and Community Detection

From our subgraphs of communities, we can detect cliques. A clique represents a densely connected group of users. We can use this to take a deeper look into a popular Yelper’s network to visualize their sphere of influence.

The size of each node (user) indicates their connections (number of friends). Each edge (link) shows connections between nodes (user’s friends). Michelle and Bryant appear twice because it shows that the location of their influence occurs in more than one group.

### Betweenness Centrality

In the clique above, we found that Paige has the highest betweenness centrality — a measure of how many times a node (user) acts as a bridge between two nodes.

Bridges are important because they connect two different groups in a social network; they are useful in bridging the gap between different communities. An example of a bridge is someone who is able to communicate and interpret data to both tech and non-tech team members. Another example is when your friend introduces you to their new friend, with your friend acting as the bridge.

## Code

# R Code for Social Network Analysis and Community Detection

# Install and load necessary libraries

#install.packages("igraph") # This is likely already installed in the environment, no need to reinstall

library(igraph)

# Install and load the 'igraphdata' package

if (!requireNamespace("igraphdata", quietly = TRUE)) {

  install.packages("igraphdata")

}

library(igraphdata)

# Load the karate club dataset

data(karate)

# Apply the Louvain community detection algorithm

community\_louvain <- cluster\_louvain(karate)

# Print the communities detected

print(community\_louvain)

# Plot the network with communities

plot(community\_louvain, karate, main="Karate Club Network with Louvain Communities")

# Calculate Degree Centrality

degree\_centrality <- degree(karate)

# Calculate Betweenness Centrality

betweenness\_centrality <- betweenness(karate)

# Calculate Closeness Centrality

closeness\_centrality <- closeness(karate)

# Print top 5 nodes with highest degree centrality

print("Top 5 nodes by degree centrality:")

print(head(sort(degree\_centrality, decreasing=TRUE), 5))

# Print top 5 nodes by betweenness centrality

print("Top 5 nodes by betweenness centrality:")

print(head(sort(betweenness\_centrality, decreasing=TRUE), 5))

# Print top 5 nodes by closeness centrality

print("Top 5 nodes by closeness centrality:")

print(head(sort(closeness\_centrality, decreasing=TRUE), 5))

# Visualize the network with node size based on degree centrality

plot(karate, vertex.size=degree\_centrality \* 2, main="Karate Club Network (Degree Centrality)")

# Find nodes with high betweenness centrality

high\_betweenness <- which(betweenness\_centrality > 10)

# Plot the network, highlighting nodes with high betweenness centrality

plot(karate,

     vertex.size=degree\_centrality \* 2,

     vertex.color=ifelse(1:vcount(karate) %in% high\_betweenness, "red", "lightblue"),

     main="Karate Club Network with High Betweenness Nodes")

# Detect cliques in the network

cliques <- cliques(karate)

# Print the largest cliques

print("Largest cliques in the network:")

print(cliques)

# Visualize the network, highlighting cliques

plot(karate,

     vertex.size=degree\_centrality \* 2,

     vertex.color=ifelse(1:vcount(karate) %in% unlist(cliques), "green", "lightblue"),

     main="Karate Club Network with Highlighted Cliques")

# Create a hypothetical attribute for joining years (for demonstration)

V(karate)$join\_year <- sample(c(2005, 2015), vcount(karate), replace=TRUE)

# Subset the network for users who joined in 2005

# Use induced\_subgraph instead of subgraph.vertices

karate\_2005 <- induced\_subgraph(karate, V(karate)[join\_year == 2005])

# Subset the network for users who joined in 2015

# Use induced\_subgraph instead of subgraph.vertices

karate\_2015 <- induced\_subgraph(karate, V(karate)[join\_year == 2015])

# Community detection for users who joined in 2005

community\_2005 <- cluster\_louvain(karate\_2005)

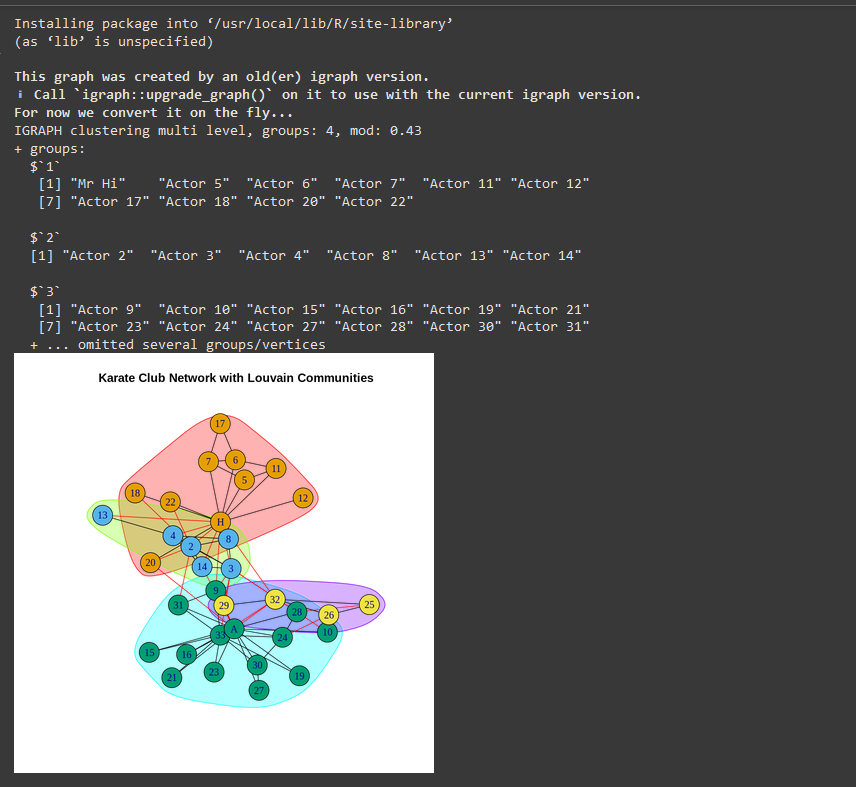
plot(community\_2005, karate\_2005, main="Community Detection (2005)")

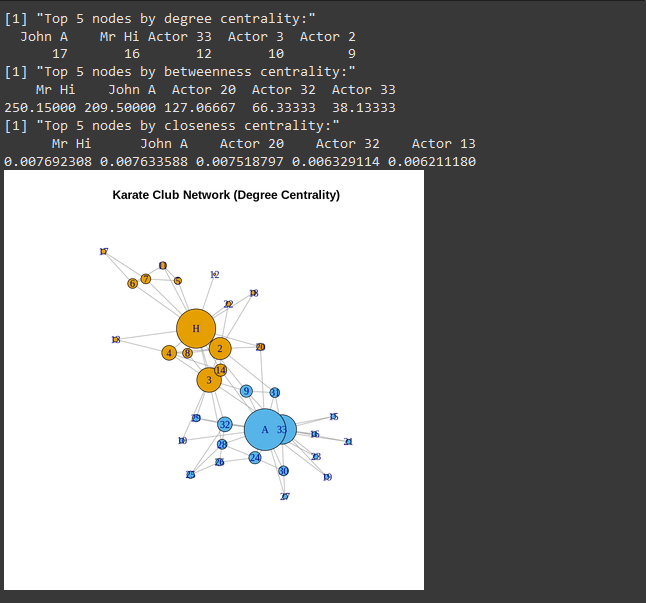
# Community detection for users who joined in 2015

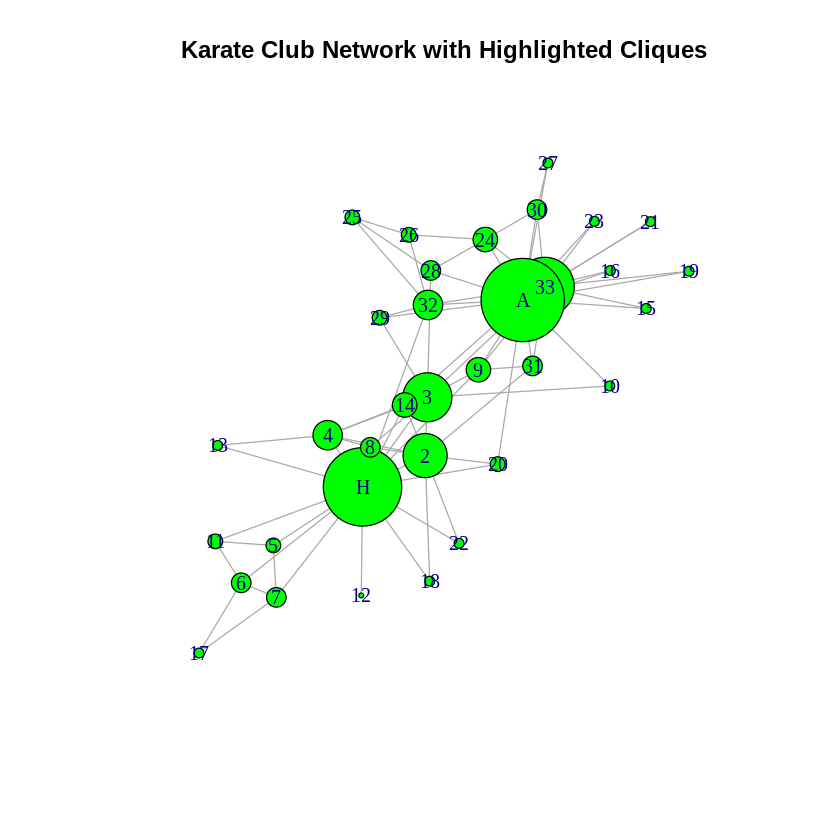
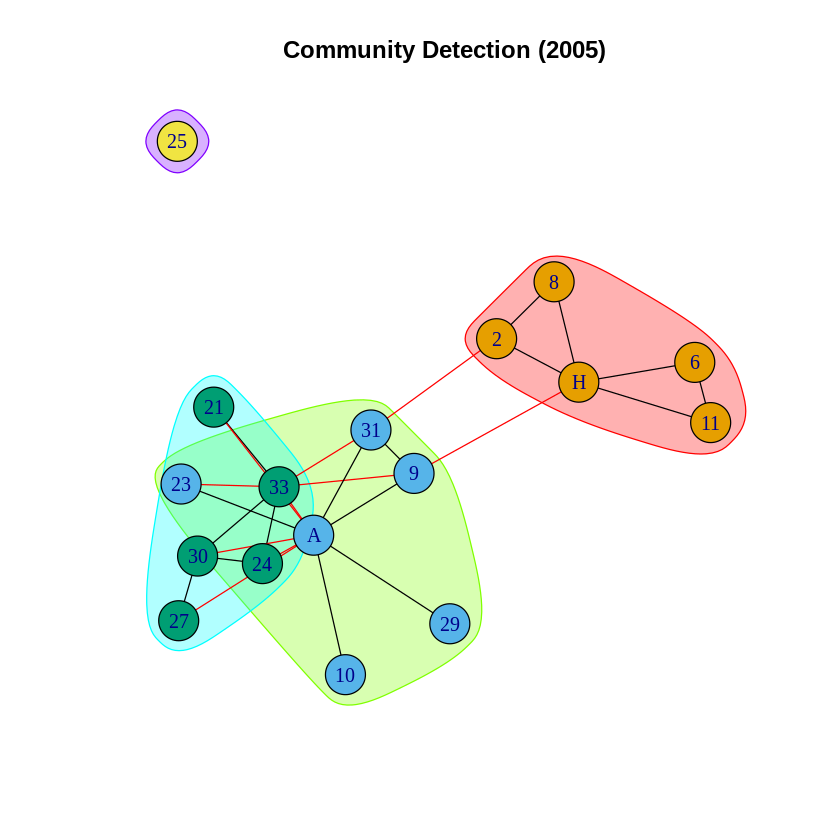
community\_2015 <- cluster\_louvain(karate\_2015)

plot(community\_2015, karate\_2015, main="Community Detection (2015)")

## Output Screenshot







## Conclusion

Social Network Analysis was successfully implemented in R, and the community detection algorithm effectively identified distinct communities within the network. The Louvain method was able to group nodes that were more densely connected to each other than to the rest of the network. The results provide valuable insights into the structure and connections within the network, as well as the identification of key nodes (users) with a significant influence in the network through betweenness centrality. This analysis helps in understanding the patterns of social connections and influence in large-scale networks.