

**MIT Art Design and Technology University
MIT School of Computing, Pune**

**Department of Computer Science and
Engineering**

Lab Manual

Practical – BDA Lab

Class - LY AIA (Sem - I)

Name of the Practical Coordinator

Dr.Nandkishor Karlekar

Team Members

1. Prof.Arati Pimpalkar
2. Prof. Hemant Shinde
3. Dr.Amreen Khan
4. Dr.Amupama Budhewar

A.Y. 2024 - 2025

INDEX

Expt. No.	Title
1	Getting data to work with: Download the sample dataset locally for any application (Kaggle) ii. Setting up the working directory. iii. Unpacking the data. Decompress the file locally. iv. Looking at the data. Display the top (10) and bottom (10) of the file. v. Measuring the length of the data set. Count the number of lines in the file. vi. Encode the categorical data vi. Plot a graph and give your insights for the application selected cases.
2	I. Perform text analysis using R/ Python. II. Perform data analysis using R programming. III. Data visualization using R Programming (Consider different input files like csv, excel, JSON etc.)
3	Installation of Apache Spark.Writing a program in Spark for Word Count with Unit Tests: Change Application
4	PySpark - Read CSV file into Data Frame Create and query a HIVE table in PySpark.
5	To implement the K-means algorithm and show the results graphically (using R programming). (Here the students are expected to use R programming and comment on the value of k selected for the application and analyze the output. (why and how the k value is selected))
6	Compute TF-IDF values of words i) From a corpus having unique values. ii) From a corpus having similar documents iii) A single word repeated multiple times in multiple documents.
7	Analytical representation of Linear regression using Movie recommendation dataset
8	Data streaming and pipelining using online cloud platform using confluent for Kafka cloud
9	Social Network Analysis using R (for example: Community Detection Algorithm)
10	Mini Project: One real life large data application to be implemented (Use standard Datasets available on the web). - Streaming data analysis – use flume for data capture, HIVE/PYSpark for analysis of twitter data, chat data, weblog analysis etc. - Recommendation System (for example: Health Care System, Stock Market Prediction, Movie Recommendation, etc.) Spatio Temporal Data Analytics

EXPERIMENT NO. 01

- i. Getting data to work with: Download the sample dataset locally for any application (Kaggle)
- ii. Setting up the working directory.
- iii. Unpacking the data. Decompress the file locally.
- iv. Looking at the data. Display the top (10) and bottom (10) of the file.
- v. Measuring the length of the data set. Count the number of lines in the file.
- vi. Encode the categorical data
- vi. Plot a graph and give your insights for the application selected cases.

Link for dataset <https://www.kaggle.com/datasets/tomaslui/healthcare-dataset?resource=download>

Setting Up the Working Directory

To set the working directory in R, you can use the `setwd()` function. Make sure to provide the path to your desired directory. Here's an example:

```
r
Copy code
# Replace 'path_to_your_directory' with your actual directory path
setwd("path_to_your_directory")
```

Alternatively, if you prefer a more interactive way, you can use RStudio's interface to set the working directory:

1. Go to **Session** in the top menu.
2. Select **Set Working Directory**.
3. Choose **Choose Directory...** and navigate to the directory you want.

Unpacking Data (Decompressing Files)

To decompress files in R, you typically use functions that correspond to the compression format of your file. Here are examples for common compression formats:

Decompressing ZIP Files

For ZIP files, you can use the `unzip()` function in R:

```
r
Copy code
# Replace 'file.zip' with the name of your ZIP file
unzip("file.zip", exdir = "output_directory")
```

- `file.zip` is the name of your ZIP file.

- `exdir = "output_directory"` specifies the directory where you want to extract the contents. Replace `"output_directory"` with your desired output directory path.

Decompressing GZIP Files

For GZIP files, you can use the `untar()` function in R:

```
r
Copy code
# Replace 'file.tar.gz' with the name of your GZIP file
untar("file.tar.gz", exdir = "output_directory")
```

- `file.tar.gz` is the name of your GZIP file.
- `exdir = "output_directory"` specifies the directory where you want to extract the contents. Replace `"output_directory"` with your desired output directory path.

Example Workflow

Here's a complete example that sets the working directory and decompresses a ZIP file:

```
r
Copy code
# Set the working directory
setwd("path_to_your_directory")

# Unzip the file
unzip("data.zip", exdir = "data")
```

Replace `"path_to_your_directory"` with your actual directory path, `"data.zip"` with your actual ZIP file name, and `"data"` with the directory where you want to extract the contents.

Notes

- Make sure you have appropriate permissions to write to the specified directory.
- Adjust the file names and paths according to your specific case.
- Depending on your system and R setup, you may need additional packages for handling specific compression formats not covered here (like `.tar.gz` or `.7z`).

To achieve tasks related to inspecting data in R, such as displaying the top and bottom rows of a file and measuring the length (number of lines) of the dataset, you can use the following R code snippets.

Loading and Viewing Data

Assume you have a CSV file named `data.csv` that you want to inspect.

```
r
Copy code
# Set the working directory if not already set
setwd("path_to_your_directory")

# Read the CSV file into a data frame
data <- read.csv("data.csv")

# Display the top 10 rows
```

```
head(data, 10)

# Display the bottom 10 rows
tail(data, 10)
```

- `read.csv("data.csv")` reads the CSV file `data.csv` into a data frame named `data`.
- `head(data, 10)` displays the top 10 rows of the data frame `data`.
- `tail(data, 10)` displays the bottom 10 rows of the data frame `data`.

Measuring the Length of the Dataset

To count the number of lines (observations) in the dataset:

```
r
Copy code
# Count the number of rows (length of the dataset)
num_rows <- nrow(data)
print(num_rows)
```

- `nrow(data)` returns the number of rows (observations) in the data frame `data`.
- `print(num_rows)` prints the number of rows to the console.

Example Workflow

Here's a combined example that demonstrates how to load a CSV file, display its top and bottom rows, and measure its length:

```
r
Copy code
# Set the working directory
setwd("path_to_your_directory")

# Read the CSV file into a data frame
data <- read.csv("data.csv")

# Display the top 10 rows
cat("Top 10 rows:\n")
print(head(data, 10))

# Display the bottom 10 rows
cat("\nBottom 10 rows:\n")
print(tail(data, 10))

# Count the number of rows (length of the dataset)
num_rows <- nrow(data)
cat("\nNumber of rows in the dataset:", num_rows, "\n")
```

Replace `"path_to_your_directory"` with the actual directory path where your `data.csv` file is located.

Notes

- Ensure that your CSV file (`data.csv` in this example) is in the specified directory and that you have permission to read from that directory.
- Adjust the number 10 in `head(data, 10)` and `tail(data, 10)` to display more or fewer rows as needed.

- This approach assumes your data is in CSV format. If your data is in a different format (e.g., Excel, text files), you'll need to use corresponding functions (`read.xlsx`, `read.table`, etc.) to read the data into R.

R language program for Encode the categorical data vi. Plot a graph and give your insights for the application selected cases.

ChatGPT

ChatGPT

To encode categorical data and then plot a graph for insights, we'll go through the steps using R. Let's assume you have a dataset with categorical variables that you want to encode and visualize. Here's a step-by-step approach:

Step 1: Loading and Preparing the Data

Assume you have a dataset `data.csv` that includes categorical variables. We will use the `read.csv()` function to read the data into R.

```
r
Copy code
# Set the working directory
setwd("path_to_your_directory")

# Read the CSV file into a data frame
data <- read.csv("data.csv", stringsAsFactors = TRUE) # Ensure strings are read as
factors for encoding
```

Step 2: Encoding Categorical Variables

To encode categorical variables, we will use the `factor()` function in R. If your data is already read with `stringsAsFactors = TRUE`, factors are automatically created. Otherwise, you can explicitly convert character columns to factors.

```
r
Copy code
# Check the structure of your data
str(data)

# If needed, convert character columns to factors
# Example:
# data$Category <- factor(data$Category)
```

Step 3: Plotting a Graph and Providing Insights

Now, let's plot a graph to visualize insights from the encoded categorical data. Depending on your data and the insights you want to gain, you can choose appropriate types of plots such as bar plots, pie charts, etc.

Here's an example using a bar plot to show the distribution of a categorical variable `Category`:

```
r
Copy code
# Load ggplot2 library for plotting
library(ggplot2)
```

```
# Plot a bar plot of Category
ggplot(data, aes(x = Category)) +
  geom_bar(fill = "blue") +
  labs(title = "Distribution of Categories",
        x = "Category",
        y = "Count") +
  theme_minimal()
```

Example Workflow

Assuming `Category` is a categorical variable in your dataset:

```
r
Copy code
# Example data loading and encoding
setwd("path_to_your_directory")
data <- read.csv("data.csv", stringsAsFactors = TRUE)

# Plotting a bar plot for Category
library(ggplot2)
ggplot(data, aes(x = Category)) +
  geom_bar(fill = "blue") +
  labs(title = "Distribution of Categories",
        x = "Category",
        y = "Count") +
  theme_minimal()
```

Replace `"path_to_your_directory"` with the actual path where your `data.csv` file is located. Ensure your dataset contains a column named `Category` or adjust the plot code accordingly based on your actual data structure.

Insights

- **Visualization:** The bar plot visualizes the distribution of categories within your dataset, allowing you to see the frequency or proportion of each category.
- **Interpretation:** From the plot, you can identify which categories are most common or least common, which may provide insights into the characteristics of your data.

Notes

- Depending on your specific dataset and research questions, you might need to customize the plot further or use different types of plots (e.g., pie chart, stacked bar chart) for better visualization.
- Ensure to install and load necessary packages (`ggplot2` in this case) using `install.packages("ggplot2")` and `library(ggplot2)` respectively, if you haven't already.

Experiment no.2-1

AIM: Perform text analysis using R/ Python.

Theory:

R Programming Language is used for statistical computing and is used by many data miners and statisticians for developing statistical software and data analysis. It includes machine learning algorithms, linear regression, time series, and statistical inference to name a few. R and its libraries implement a wide variety of statistical and graphical techniques, including linear and non-linear modeling, classical, statistical tests, time-series analysis, classification, clustering, and others.

Any value written inside the double quote is treated as a string in R. String is an array of characters and these collections of characters are stored inside a variable. Internally R stores every string within double quotes, even when you create them with a single quote.

Text Processing in R

Using Built-in Type in R

Using Tidyverse module

Using regex and external module

Using grep()

Text analysis (TA) is a machine learning technique used to automatically extract valuable insights from unstructured text data. Companies use text analysis tools to quickly digest online data and documents, and transform them into actionable insights.

You can use text analysis to extract specific information, like keywords, names, or company information from thousands of emails, or categorize survey responses by sentiment and topic.

When you put machines to work on organizing and analyzing your text data, the insights and benefits are huge.

Steps involved:

Step 1: Import dataset with setting delimiter

Step 2: Text Cleaning or Preprocessing

- Remove Punctuations, Numbers

- Stemming

- Convert each word into its lower case

Step 3: Tokenization, involves splitting sentences and words from the body of the text.

Step 4: Making the bag of words and analyse the final result.

CODE:

To be executed in R tool with the data set named: "TeamHealthRawDataForDemo". Lessen the number of lines for quick execution.

```
install.packages("tm") # for text mining
```

```
install.packages("SnowballC") # for text stemming
```

```
install.packages("wordcloud") # word-cloud generator
```

```
install.packages("RColorBrewer") # color palettes
```

```
install.packages("syuzhet") # for sentiment analysis
```

```
install.packages("ggplot2") # for plotting graphs
```

```
# Load
```

```
library("tm")
```

```
library("SnowballC")
```



```

library("wordcloud")
library("RColorBrewer")
library("syuzhet")
library("ggplot2")

text <- readLines(file.choose())
# Load the data as a corpus
TextDoc <- Corpus(VectorSource(text))

#Replacing "/", "@" and "|" with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
TextDoc <- tm_map(TextDoc, toSpace, "/")
TextDoc <- tm_map(TextDoc, toSpace, "@")
TextDoc <- tm_map(TextDoc, toSpace, "\\|")
# Convert the text to lower case
TextDoc <- tm_map(TextDoc, content_transformer(tolower))
# Remove numbers
TextDoc <- tm_map(TextDoc, removeNumbers)
# Remove english common stopwords
TextDoc <- tm_map(TextDoc, removeWords, stopwords("english"))
# Remove your own stop word
# specify your custom stopwords as a character vector
TextDoc <- tm_map(TextDoc, removeWords, c("s", "company", "team"))
# Remove punctuations
TextDoc <- tm_map(TextDoc, removePunctuation)
# Eliminate extra white spaces
TextDoc <- tm_map(TextDoc, stripWhitespace)
# Text stemming - which reduces words to their root form
TextDoc <- tm_map(TextDoc, stemDocument)

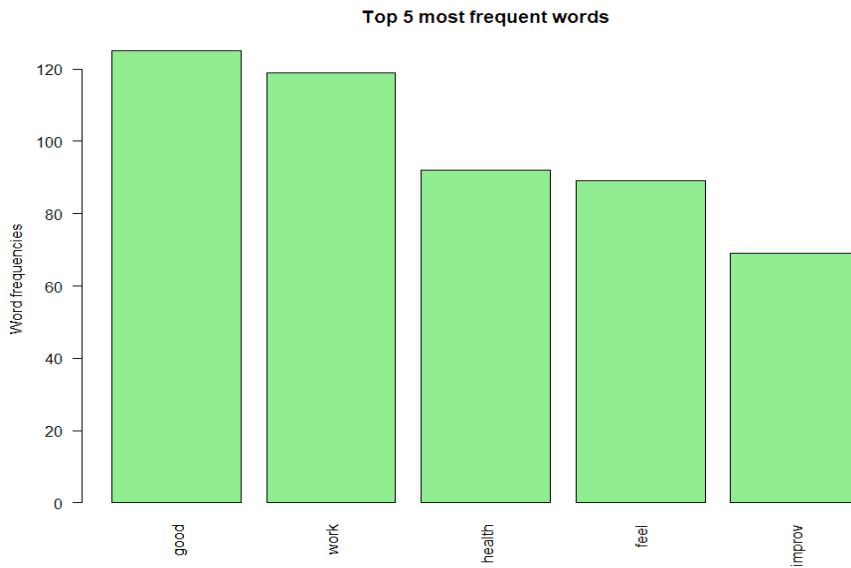
# Build a term-document matrix
TextDoc_dtm <- TermDocumentMatrix(TextDoc)
dtm_m <- as.matrix(TextDoc_dtm)
# Sort by decreasing value of frequency
dtm_v <- sort(rowSums(dtm_m),decreasing=TRUE)
dtm_d <- data.frame(word = names(dtm_v),freq=dtm_v)
# Display the top 5 most frequent words
head(dtm_d, 5)

# Plot the most frequent words
barplot(dtm_d[1:5,]$freq, las = 2, names.arg = dtm_d[1:5,]$word,
        col = "lightgreen", main = "Top 5 most frequent words",
        ylab = "Word frequencies")

#generate word cloud
set.seed(1234)
wordcloud(words = dtm_d$word, freq = dtm_d$freq, min.freq = 5,
          max.words=100, random.order=FALSE, rot.per=0.40,

```

Output:



Experiment No: 2-2

AIM: Perform data analysis using R programming

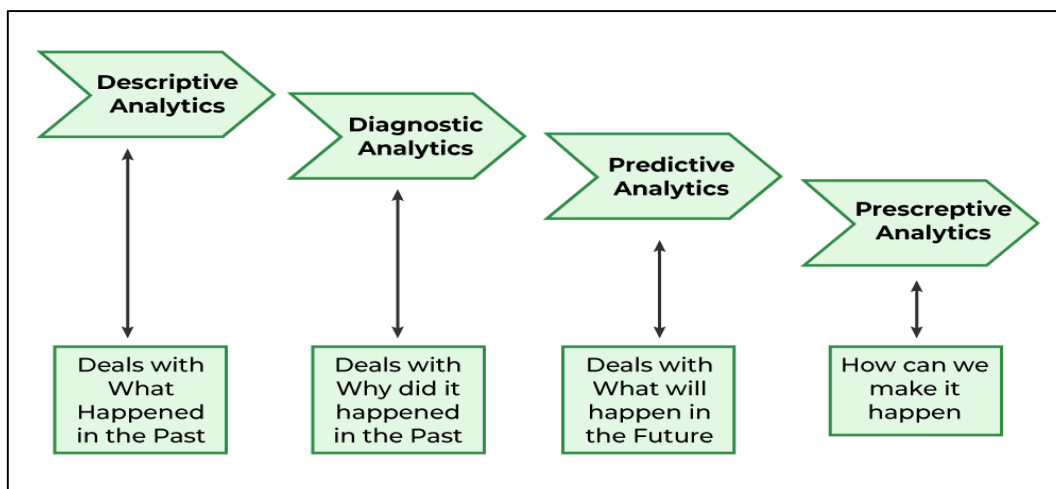
THEORY:

Data analysis using R: Data Analysis is a subset of data analytics, it is a process where the objective has to be made clear, collect the relevant data, preprocess the data, perform analysis (understand the data, explore insights), and then visualize it. The last step visualization is important to make people understand what's happening in the firm.

Types of Data Analytics

There are four major types of data analytics:

1. Predictive (forecasting)
2. Descriptive (business intelligence and data mining)
3. Prescriptive (optimization and simulation)
4. Diagnostic analytics



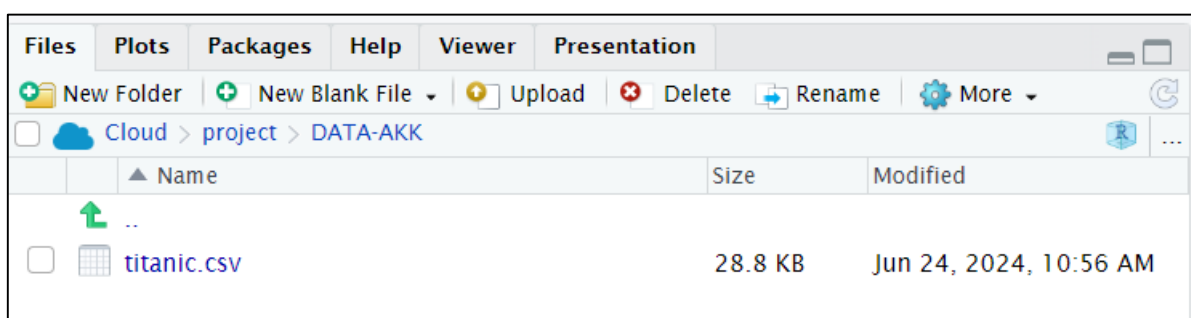
Steps in Data Analysis



Data Analysis using the Titanic dataset:

Save the dataset in the current working directory, now we will start analysis (getting to know our data).

Students can prefer the Free Posit Cloud. Posit Cloud (formerly RStudio Cloud) lets you access Posit's powerful set of data science tools right in your browser – no installation or complex configuration required. And can choose to sign in for free. (read site instructions carefully)



```
titanic=read.csv("train.csv")
```

```
head(titanic)
```

```
> titanic=read.csv("titanic.csv")
> head(titanic)
  PassengerId Survived Pclass                    Name Sex
1          892         0      3                Kelly, Mr. James male
2          893         1      3   Wilkes, Mrs. James (Ellen Needs) female
3          894         0      2         Myles, Mr. Thomas Francis male
4          895         0      3                Wirz, Mr. Albert male
5          896         1      3 Hirvonen, Mrs. Alexander (Helga E Lindqvist) female
6          897         0      3   Svensson, Mr. Johan Cervin male
  Age SibSp Parch Ticket   Fare Cabin Embarked
1 34.5     0     0 330911  7.8292      Q
2 47.0     1     0 363272  7.0000      S
3 62.0     0     0 240276  9.6875      Q
4 27.0     0     0 315154  8.6625      S
5 22.0     1     1 3101298 12.2875      S
6 14.0     0     0   7538  9.2250      S
```

To understand the class(data type) of each column **sapply()** method can be used.

```
sapply(titanic,class)
```

```
> sapply(titanic, class)
PassengerId      Survived      Pclass      Name      Sex      Age      SibSp
"integer"      "integer"      "integer" "character" "character" "numeric" "integer"
      Parch      Ticket      Fare      Cabin      Embarked
"integer" "character" "numeric" "character" "character"
> |
```

To analyze data using a summary of all the columns, their values, and data types. **summary()** can be used for this purpose.

```
summary(titanic)
```

```
> summary(titanic)
 PassengerId      Survived      Pclass      Name
Min.   : 892.0   Min.   :0.0000   Min.   :1.000   Length:418
1st Qu.: 996.2   1st Qu.:0.0000   1st Qu.:1.000   Class :character
Median :1100.5   Median :0.0000   Median :3.000   Mode  :character
Mean   :1100.5   Mean   :0.3636   Mean   :2.266
3rd Qu.:1204.5   3rd Qu.:1.0000   3rd Qu.:3.000
Max.   :1309.0   Max.   :1.0000   Max.   :3.000

      Sex      Age      SibSp      Parch
Length:418   Min.   : 0.17   Min.   :0.0000   Min.   :0.0000
Class :character 1st Qu.:21.00 1st Qu.:0.0000 1st Qu.:0.0000
Mode  :character Median :27.00 Median :0.0000 Median :0.0000
                  Mean   :30.27 Mean   :0.4474 Mean   :0.3923
                  3rd Qu.:39.00 3rd Qu.:1.0000 3rd Qu.:0.0000
                  Max.   :76.00 Max.   :8.0000 Max.   :9.0000
                  NA's   :86

      Ticket      Fare      Cabin      Embarked
Length:418   Min.   : 0.000   Length:418   Length:418
Class :character 1st Qu.: 7.896   Class :character  Class :character
Mode  :character Median :14.454   Mode  :character  Mode  :character
                  Mean   :35.627
                  3rd Qu.:31.500
                  Max.   :512.329
                  NA's   :1
```

From the above summary Students to extract below observations:

- Total passengers: 891
- The number of total people who survived: 342

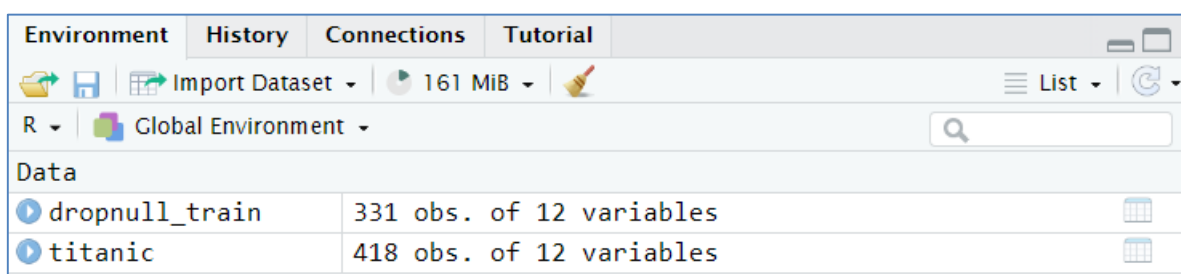
- Number of total people dead: 549
- Number of males in the titanic: 577
- Number of females in the titanic: 314
- Maximum age among all people in titanic: 80
- Median age: 28

Preprocessing of the data is important before analysis, so null values have to be checked and removed.

```
sum(is.na(train))
```

```
dropnull_train=titanic[rowSums(is.na(titanic))<=0,]
```

- dropnull_train contains only 331 rows because (total rows in dataset (418) – null value rows (87) = remaining rows (331))
- Now lets will divide survived and dead people into a separate list from 331 rows.

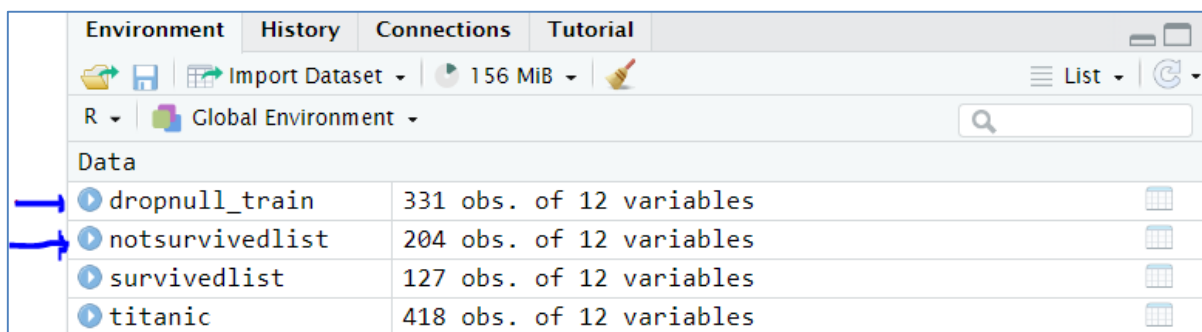


Environment	History	Connections	Tutorial
<div> <div>Import Dataset</div> <div>161 MiB</div> <div>List</div> </div> <div>R Global Environment</div> <div> <div>Search</div> </div>			
Data			
dropnull_train	331 obs. of 12 variables		
titanic	418 obs. of 12 variables		

```
survivedlist=dropnull_train[dropnull_train$Survived == 1,]
```

```
notsurvivedlist=dropnull_train[dropnull_train$Survived == 0,]
```

```
> survivedlist=dropnull_train[dropnull_train$Survived == 1,]
> notsurvivedlist=dropnull_train[dropnull_train$Survived == 0,]
> |
```



Environment	History	Connections	Tutorial
<div> <div>Import Dataset</div> <div>156 MiB</div> <div>List</div> </div> <div>R Global Environment</div> <div> <div>Search</div> </div>			
Data			
dropnull_train	331 obs. of 12 variables		
notsurvivedlist	204 obs. of 12 variables		
survivedlist	127 obs. of 12 variables		
titanic	418 obs. of 12 variables		

Visualization:

Now to visualize the number of males and females dead and survived using bar plots, histograms, and piecharts.

Bar charts are a popular and effective way to visually represent categorical data in a structured manner. R stands out as a powerful programming language for data analysis and visualization.

A bar chart also known as bar graph is a pictorial representation of data that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. In other words, it is the pictorial representation of the dataset. These data sets contain the numerical values of variables that represent the length or height.

R uses the `barplot()` function to create bar charts. Here, both vertical and Horizontal bars can be drawn.

Syntax: `barplot(H, xlab, ylab, main, names.arg, col, horiz = TRUE)`

Parameters:

H: This parameter is a vector or matrix containing numeric values which are used in bar chart.

xlab: This parameter is the label for x axis in bar chart.

ylab: This parameter is the label for y axis in bar chart.

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

names.arg: This parameter is a vector of names appearing under each bar in bar chart.

col: This parameter is used to give colors to the bars in the graph.

horizontal = TRUE

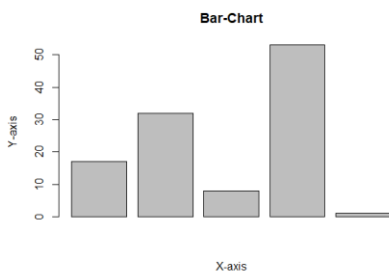
Ex:

```
# Create the data for the chart
```

```
A <- c(17, 32, 8, 53, 1)
```

```
# Plot the bar chart
```

```
barplot(A, xlab = "X-axis", ylab = "Y-axis", main = "Bar-Chart")
```



A **pie chart** is a circular statistical graphic, which is divided into slices to illustrate numerical proportions. It depicts a special chart that uses “pie slices”, where each sector shows the relative sizes of data. A circular chart cuts in the form of radii into segments describing relative frequencies or magnitude also known as a circle graph. R Programming Language uses the function `pie()` to create pie charts. It takes positive numbers as a vector input.

Syntax: `pie(x, labels, radius, main, col, clockwise)`

Parameters:

x: This parameter is a vector that contains the numeric values which are used in the pie chart.

labels: This parameter gives the description to the slices in pie chart.

radius: This parameter is used to indicate the radius of the circle of the pie chart.(value between -1 and +1).

main: This parameter is represents title of the pie chart.

clockwise: This parameter contains the logical value which indicates whether the slices are drawn clockwise or in anti clockwise direction.

col: This parameter give colors to the pie in the graph.

Ex:

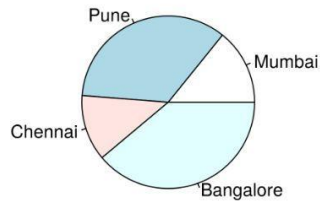
```
# Create data for the graph.
```

```
Count<- c(23, 56, 20, 63)
```

```
labels <- c("Mumbai", "Pune", "Chennai", "Bangalore")
```

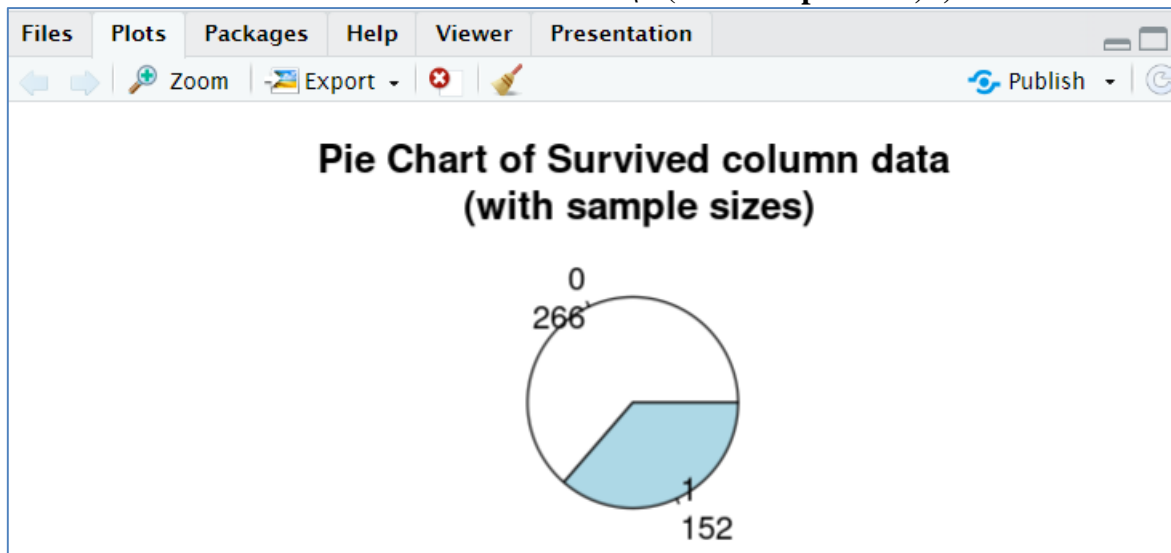
```
# Plot the chart.
```

```
pie(count, labels)
```



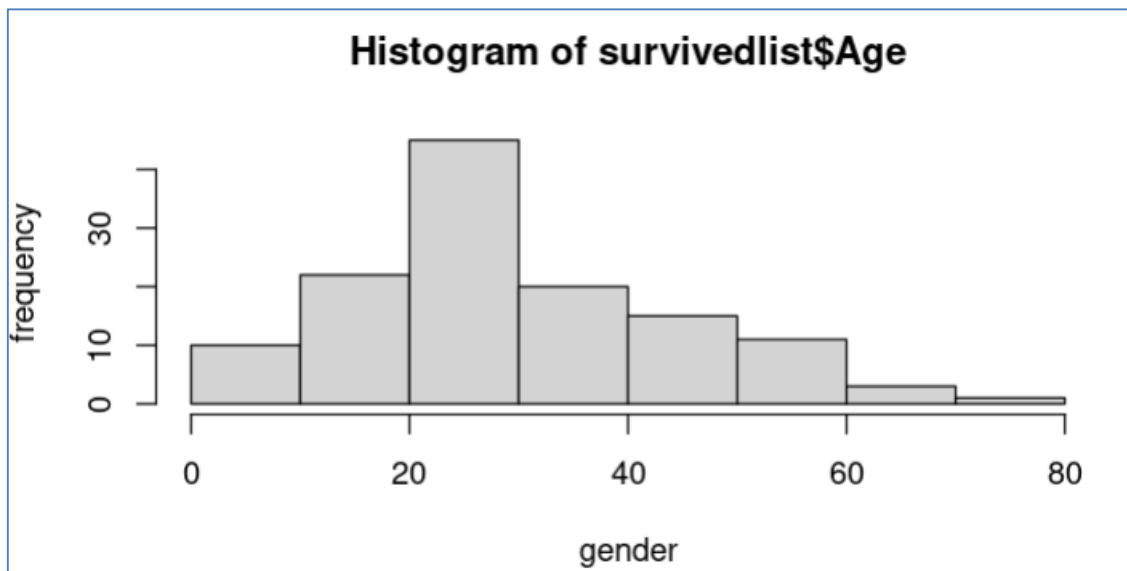
For the Titanic data set, creating a pie chart to visualize the number of males and females dead and survived.

```
mytable <- table(titanic$Survived)
lbls <- paste(names(mytable), "\n", mytable, sep="")
pie(mytable,
    labels = lbls,
    main="Pie Chart of Survived column data\n (with sample sizes)")
```



```
hist(survivedlist$Age,
     xlab="gender",
     ylab="frequency")
```

```
> hist(survivedlist$Age,
+       xlab="gender",
+       ylab="frequency")
> |
```



```
barplot(table(notsurvivedlist$Sex),  
        xlab="gender",  
        ylab="frequency")
```

Inclass Assessment:

1. Draw barplot to Analyse males and females those who not survived in titanic.
2. There are some passengers who are charged extremely high. So, these values can affect the analysis as they are outliers. Confirm their presence using a boxplot.

Experiment No: 2-3

Aim: Data visualization using R Programming (Consider different input files like csv, excel, JSON etc.)

Theory:

In R, you can read data from files outside of the R environment. One may also write data to files that the operating system can store and further access. There is a wide range of file formats, including CSV, Excel, binary, and XML, etc., R can read and write from.

While many organizations store data in databases and storage options such as AWS, Azure, and GCP, Microsoft Excel spreadsheets continue to be widely used for storing smaller datasets.

Excel's data science functionality is more limited than R's, so it's useful to be able to import data from spreadsheets to R.

We may work with structured data from spreadsheets, take advantage of R's capabilities for data analysis and manipulation, and incorporate Excel data into other R processes and packages by reading Excel files in R. The readxl package offers a simple and effective method for reading Excel files into R as data frames for additional processing and analysis.

Reading Data from Excel Files in R

These files are used to store data in a tabular form and are commonly employed in data analysis and manipulation tasks. Must have worked with structured data from spreadsheets, can now take advantage of R's capabilities for data analysis and manipulation, and incorporate Excel data into other R processes and packages by reading Excel files in R.

R provides several packages like readxl, xlsx, and openxlsx to read or import Excel files into R DataFrame. These packages provide several methods with different arguments which help us read Excel files effectively.

Two different techniques to read or import an Excel file in R.

Method 1: Using read_excel() from readxl

Method 2: Using read.xlsx() from xlsx

Approach

- Import module
- Pass the path of the file to the required function
- Read file
- Display content

Step 1: Installing Necessary Packages

Install the necessary packages

install.packages("readxl")

install.packages("writexl")

The "install.packages()" function is used to install packages in R. the "readxl" package is being installed. This package provides functions for reading Excel files into R. See below output in the console, signaling successful installation.

```
> install.packages("readxl")
Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
(as 'lib' is unspecified)
also installing the dependencies 'cli', 'glue', 'utf8', ' rematch', 'fansi', 'lifecycle',
'magrittr', 'pillar', 'pkgconfig', 'rlang', 'vctrs', 'crayon', 'hms', 'prettyunits', 'R
6', 'cellranger', 'tibble', 'cpp11', 'progress'

trying URL 'http://rspm/default/__linux__/focal/latest/src/contrib/cli_3.6.3.tar.gz'
Content type 'application/x-gzip' length 1267179 bytes (1.2 MB)
=====
downloaded 1.2 MB

trying URL 'http://rspm/default/__linux__/focal/latest/src/contrib/glue_1.7.0.tar.gz'
Content type 'application/x-gzip' length 149591 bytes (146 KB)
=====
downloaded 146 KB
```

Step 2: Loading Packages

Load the necessary packages

```
library(readxl)
library(writexl)
```

The code loads the readxl package in R. The library() function is used to load packages in R, which are collections of functions and data sets that extend the functionality of R. The readxl package provides functions for reading Excel files into R. By loading this package, the user can access these functions and use them in their R code.

```
The downloaded source packages are in
      '/tmp/RtmpHL67Be/downloaded_packages'
> library(readxl)
> |
```

Step 3: Reading an Excel File

Use the function read_excel() from the 'readxl' package. This function requires as an argument the path to the Excel file.

Read an Excel file

```
data <- read_excel("path/to/your/file.xlsx")
```

```
iris <- read_xlsx("sample-dataset 2-3.xlsx", sheet = "iris")
```

OR

```
iris2 <- read_xlsx("sample-dataset 2-3.xlsx", sheet = 1)
```

The code reads an Excel file named "sample.xlsx" and extracts the data from the sheet named "iris". The data is then stored in a data frame named "iris". The "<" symbol is an HTML entity that represents the less than sign "<". In R, the less than sign is used for assignment, so this code assigns the data from the Excel sheet to the "iris" data frame.

```
> iris
# A tibble: 150 × 6
   Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species
  <dbl>         <dbl>         <dbl>         <dbl>         <dbl> <chr>
1     1           5.1           3.5           1.4           0.2 Iris-setosa
2     2           4.9           3           1.4           0.2 Iris-setosa
3     3           4.7           3.2           1.3           0.2 Iris-setosa
4     4           4.6           3.1           1.5           0.2 Iris-setosa
5     5           5           3.6           1.4           0.2 Iris-setosa
6     6           5.4           3.9           1.7           0.4 Iris-setosa
7     7           4.6           3.4           1.4           0.3 Iris-setosa
8     8           5           3.4           1.5           0.2 Iris-setosa
9     9           4.4           2.9           1.4           0.2 Iris-setosa
10    10           4.9           3.1           1.5           0.1 Iris-setosa
# i 140 more rows
```

About the dataset used:

The dataset read into R is a small one with only two sheets to demonstrate how to specify which sheet to read. The first sheet is a bank marketing dataset with 45,211 rows and 17 columns. The screenshot below is from the excel file "sample-dataset 2-3.xlsx" and sheet name "bank-full".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
2	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
3	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
4	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
5	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
6	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no
7	35	management	married	tertiary	no	231	yes	no	unknown	5	may	139	1	-1	0	unknown	no
8	28	management	single	tertiary	no	447	yes	yes	unknown	5	may	217	1	-1	0	unknown	no
9	42	entrepreneur	divorced	tertiary	yes	2	yes	no	unknown	5	may	380	1	-1	0	unknown	no
10	58	retired	married	primary	no	121	yes	no	unknown	5	may	50	1	-1	0	unknown	no
11	43	technician	single	secondary	no	593	yes	no	unknown	5	may	55	1	-1	0	unknown	no
12	41	admin.	divorced	secondary	no	270	yes	no	unknown	5	may	222	1	-1	0	unknown	no
13	29	admin.	single	secondary	no	390	yes	no	unknown	5	may	137	1	-1	0	unknown	no
14	53	technician	married	secondary	no	6	yes	no	unknown	5	may	517	1	-1	0	unknown	no
15	58	technician	married	unknown	no	71	yes	no	unknown	5	may	71	1	-1	0	unknown	no
16	57	services	married	secondary	no	162	yes	no	unknown	5	may	174	1	-1	0	unknown	no

The second sheet is the Iris dataset with 150 rows and 6 columns and contains information about Iris flower types, such as their sepal and petal lengths and widths. The screenshot below is from the same excel file, “sample.xlsx” and sheet name “iris”.

	A	B	C	D	E	F
1	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa

Environment	History	Connections	Tutorial
<div> <div> <div>Import Dataset</div> <div>588 MiB</div> <div>List</div> </div> <div> <div>R</div> <div>Global Environment</div> <div></div> </div> </div>			
Data			
bank_df_s2		45209 obs. of 17 variables	
iris		150 obs. of 6 variables	

Reading Specific Rows

Let’s read specific rows from a workbook by setting the skip and n_max arguments. For skipping the first few rows, you can use the skip argument with a value equal to the number of rows you want to skip.

```
bank_df_s2 <- read_excel("sample-dataset 2-3.xlsx", sheet = "bank-full", skip = 2)
```

Please note that the above code skips the headers as well.

```
> bank_df_s2
# A tibble: 45,209 × 17
  `44.0` technician single secondary no...5 `29.0` yes no...8 unknown...9 `5.0`
  <dbl> <chr> <chr> <chr> <chr> <dbl> <chr> <chr> <chr> <dbl>
1 33 entrepreneur married secondary no 2 yes yes unknown 5
2 47 blue-collar married unknown no 1506 yes no unknown 5
3 33 unknown single unknown no 1 no no unknown 5
4 35 management married tertiary no 231 yes no unknown 5
5 28 management single tertiary no 447 yes yes unknown 5
6 42 entrepreneur divorc... tertiary yes 2 yes no unknown 5
7 58 retired married primary no 121 yes no unknown 5
8 43 technician single secondary no 593 yes no unknown 5
9 41 admin. divorc... secondary no 270 yes no unknown 5
10 29 admin. single secondary no 390 yes no unknown 5
# i 45,199 more rows
# i 7 more variables: may <chr>, `151.0` <dbl>, `1.0` <dbl>, `-1.0` <dbl>,
```

Step 4: Viewing the Data

After reading an Excel file, you probably want to view the data. You can do this using the `print()` function.

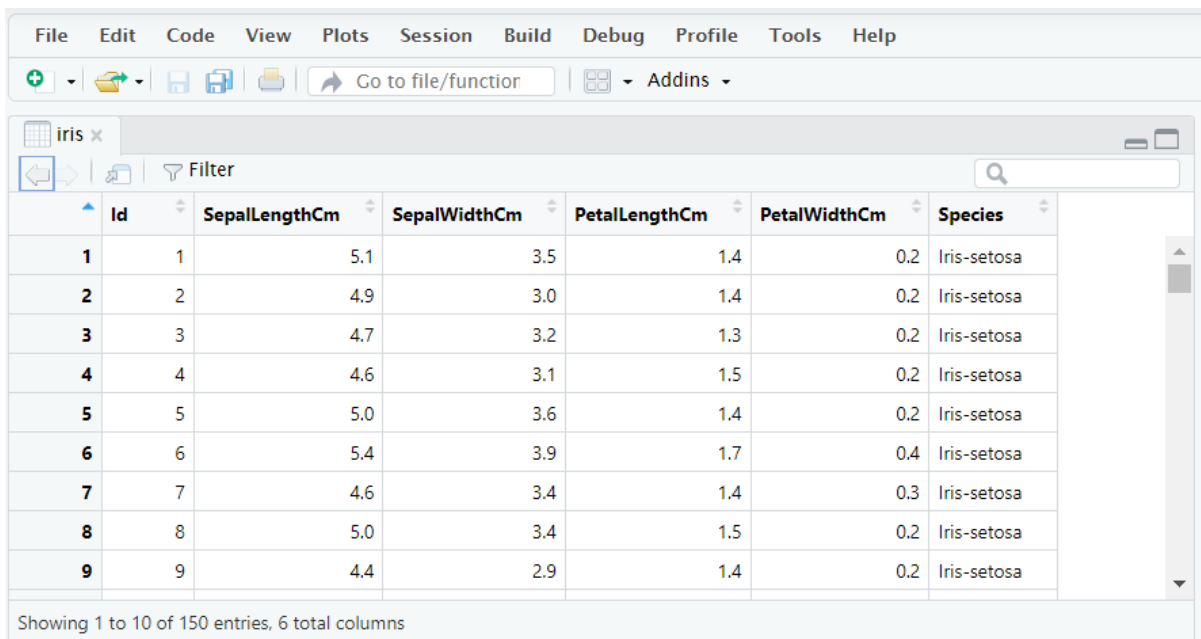
```
# Print the data
```

```
print(data)
```

Alternatively, you can use the `View()` function to open your data in a spreadsheet-like format.

```
# View the data
```

```
View(data) //V is Capital
```



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	5.1	3.5	1.4	0.2	Iris-setosa
2	2	4.9	3.0	1.4	0.2	Iris-setosa
3	3	4.7	3.2	1.3	0.2	Iris-setosa
4	4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	5.0	3.6	1.4	0.2	Iris-setosa
6	6	5.4	3.9	1.7	0.4	Iris-setosa
7	7	4.6	3.4	1.4	0.3	Iris-setosa
8	8	5.0	3.4	1.5	0.2	Iris-setosa
9	9	4.4	2.9	1.4	0.2	Iris-setosa

Used-View(iris)command

Step 5: Handling Multiple Sheets

If your Excel file contains multiple sheets, you can specify the sheet you want to read using the 'sheet' argument in the `read_excel()` function.

```
# Read a specific sheet
```

```
data <- read_excel("path/to/your/file.xlsx", sheet = "Sheet2")
```

Inclass Assignment:

1. Reading Specific Cells
2. Skipping Columns

Reference Link:

<https://egyankosh.ac.in/bitstream/123456789/87562/1/Unit-14.pdf>

Experiment no. 03

Aim: a) To Install Apache Spark.

b) write a program in Spark for Word Count with Unit Tests: Change Application

Theory:

What is Apache Spark?

Apache Spark is an open-source, distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark is designed to be fast for both interactive queries and large-scale batch processing. It extends the Map Reduce model to efficiently support more types of computations, including interactive queries and stream processing.

Why Use Apache Spark?

Speed: Spark's in-memory computing capabilities make it up to 100 times faster than Hadoop MapReduce.

Ease of Use: Spark has easy-to-use APIs for operating on large datasets, which include a comprehensive set of libraries.

General-purpose: It supports various programming languages such as Java, Scala, Python, and R, making it a versatile tool.

Unified Engine: Spark provides a unified engine that can handle various types of data processing workloads, including batch processing, stream processing, and machine learning.

Prerequisites:

1. Java Development Kit (JDK): Apache Spark requires Java to be installed on your system. Ensure you have JDK 8 or later.
2. Scala: Apache Spark is written in Scala, and it's useful to have Scala installed if you want to use the Scala API.
3. Python: For using the PySpark API, ensure Python 2.7 or 3.4 and later versions are installed.

Steps to Install Apache Spark:

Step 1: Install Java Development Kit (JDK)

1. Download JDK: Go to the [Oracle JDK download page](<https://www.oracle.com/java/technologies/javase-downloads.html>) or use OpenJDK.

2. Install JDK:

- On Windows: Run the installer and follow the setup instructions.
- On macOS: Use Homebrew with the command ``brew install openjdk``.

- On Linux: Use your package manager, e.g., `sudo apt-get install openjdk-8-jdk`.

3. Set JAVA_HOME:

- On Windows: Add the JDK bin directory to the system PATH.
- On macOS/Linux: Add the following lines to your `.bashrc` or `.zshrc`:

```
```bash
export JAVA_HOME=/path/to/your/jdk
export PATH=$JAVA_HOME/bin:$PATH
```
```

Step 2: Download Apache Spark

1. Go to the Spark Download Page*: Visit the [Apache Spark download page](<https://spark.apache.org/downloads.html>).
2. Select Spark Release: Choose a Spark release version (e.g., 3.1.2) and select a pre-built package for Hadoop (e.g., Pre-built for Apache Hadoop 2.7 and later).
3. Download Spark: Download the selected Spark package and extract it to a directory of your choice.

Step 3: Set Up Environment Variables

1. Set SPARK_HOME:

- On Windows: Add `SPARK_HOME` environment variable pointing to your Spark directory.
- On macOS/Linux: Add the following lines to your `.bashrc` or `.zshrc`:

```
```bash
export SPARK_HOME=/path/to/spark
export PATH=$SPARK_HOME/bin:$PATH
```
```

2. Configure Spark:

- Copy the template configuration file:

```
```bash
cp $SPARK_HOME/conf/spark-env.sh.template $SPARK_HOME/conf/spark-env.sh
```
```

- Edit `spark-env.sh` to configure your Spark settings, such as setting `SPARK_MASTER_HOST`.

Step 4: Verify Installation

1. Start Spark Shell:

- Open a terminal and run the following command:

```
```bash
spark-shell
```
```

- You should see the Spark shell starting, and you can run Spark commands in Scala.

2. Start PySpark:

- For Python API, run:

```
```bash  
pyspark
```
```

- We should see the PySpark shell starting, allowing you to run Spark commands in Python.

b) write a program in Spark for Word Count with Unit Tests: Change Application

Theory

What is Word Count?

Word Count is a simple yet fundamental algorithm used to demonstrate the capabilities of data processing frameworks like Apache Spark. The algorithm reads a text file, splits it into words, and counts the occurrences of each word.

Why Use Spark for Word Count?

Scalability: Spark can handle large datasets distributed across a cluster of machines.

Speed: Spark's in-memory computing capabilities make it faster than traditional disk-based processing.

Ease of Use: Spark provides high-level APIs in Java, Scala, Python, and R, making it easy to implement common data processing tasks.

Spark Programming Model

Spark's programming model is based on Resilient Distributed Datasets (RDDs), which are fault-tolerant collections of elements that can be operated on in parallel. The main operations on RDDs are transformations (e.g., map, filter, reduceByKey) and actions (e.g., collect, saveAsTextFile).

Steps to Implement Word Count in Spark

Initialize Spark Context: Set up the environment to interact with Spark.

- **Read Input Data:** Load the text file into an RDD.
- **Transform Data:** Split the lines into words and map each word to a key-value pair (word, 1).
- **Aggregate Data:** Reduce the key-value pairs by key to get the word counts.
- **Save or Display Output:** Save the results to an output file or display them on the console.

Unit Testing in Spark

Unit tests are essential to ensure the correctness of your Spark application. You can use testing frameworks like unittest in Python or ScalaTest in Scala to write unit tests for your Spark application. Key aspects to test include:

Correctness of the transformation logic.

Handling of edge cases (e.g., empty lines, special characters).

Performance and scalability.

Algorithm

Word Count Algorithm in Spark

1. Initialize Spark Context:

Create a SparkContext object to interact with Spark.

2. Read Input Data:

Use the `textFile` method to load the input file into an RDD.

3. Transform Data:

Use `flatMap` to split each line into words.

Use `map` to create key-value pairs (word, 1) for each word.

4. Aggregate Data:

Use `reduceByKey` to sum the counts for each word.

5. Save or Display Output:

Use `saveAsTextFile` or `collect` to output the results.

Conclusion:

We have successfully installed the Apache Spark, creating a basic word count program in Spark, and adding unit tests to ensure the correctness of the application. With these steps, we can leverage Spark's powerful capabilities for data processing and ensure the reliability of our applications through comprehensive testing. By integrating unit tests, one can maintain code quality and quickly identify any issues, making your Spark applications robust and dependable

b) write a program in Spark for Word Count with Unit Tests: Change Application

Theory

What is Word Count?

Word Count is a simple yet fundamental algorithm used to demonstrate the capabilities of data processing frameworks like Apache Spark. The algorithm reads a text file, splits it into words, and counts the occurrences of each word.

Why Use Spark for Word Count?

Scalability: Spark can handle large datasets distributed across a cluster of machines.

Speed: Spark's in-memory computing capabilities make it faster than traditional disk-based processing.

Ease of Use: Spark provides high-level APIs in Java, Scala, Python, and R, making it easy to implement common data processing tasks.

Spark Programming Model

Spark's programming model is based on Resilient Distributed Datasets (RDDs), which are fault-tolerant collections of elements that can be operated on in parallel. The main operations on RDDs are transformations (e.g., map, filter, reduceByKey) and actions (e.g., collect, saveAsTextFile).

Steps to Implement Word Count in Spark

Initialize Spark Context: Set up the environment to interact with Spark.

- **Read Input Data:** Load the text file into an RDD.
- **Transform Data:** Split the lines into words and map each word to a key-value pair (word, 1).
- **Aggregate Data:** Reduce the key-value pairs by key to get the word counts.
- **Save or Display Output:** Save the results to an output file or display them on the console.

Unit Testing in Spark

Unit tests are essential to ensure the correctness of your Spark application. You can use testing frameworks like unittest in Python or ScalaTest in Scala to write unit tests for your Spark application. Key aspects to test include:

Correctness of the transformation logic.

Handling of edge cases (e.g., empty lines, special characters).

Performance and scalability.

Algorithm

Word Count Algorithm in Spark

6. Initialize Spark Context:

Create a `SparkContext` object to interact with Spark.

7. Read Input Data:

Use the `textFile` method to load the input file into an RDD.

8. Transform Data:

Use `flatMap` to split each line into words.

Use `map` to create key-value pairs (word, 1) for each word.

9. Aggregate Data:

Use `reduceByKey` to sum the counts for each word.

10. Save or Display Output:

Use `saveAsTextFile` or `collect` to output the results.

Conclusion:

We have successfully installed the Apache Spark, creating a basic word count program in Spark, and adding unit tests to ensure the correctness of the application. With these steps, we can leverage Spark's powerful capabilities for data processing and ensure the reliability of our applications through comprehensive testing. By integrating unit tests, one can maintain code quality and quickly identify any issues, making your Spark applications robust and

Experiment no. 04

- Aim:**
- 1) PySpark - Read CSV file into Data Frame
 - 2) Create and query a HIVE table in PySpark.

Theory:

PySpark Overview:

PySpark is the Python API for Apache Spark, an open-source, distributed computing system. It provides a programming interface for entire clusters with implicit data parallelism and fault tolerance. PySpark allows you to write Spark applications using Python.

PySpark DataFrame:

A DataFrame is a distributed collection of data organized into named columns, similar to a table in a relational database or a data frame in R or Python (with pandas). It is one of the most commonly used abstractions in Spark.

Hive and Hive Tables:

Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. It provides an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

Why Use PySpark with Hive?

- **Scalability:** Handle large datasets efficiently.
- **SQL Compatibility:** Use familiar SQL queries for data processing.
- **Integration:** Easily integrate with existing Hadoop ecosystems.

Program Details:

Prerequisites:

1. **Apache Spark:** Installed and configured.
2. **Hadoop and Hive:** Installed and configured if using a Hadoop-based deployment.
3. **Python:** Ensure Python 2.7 or 3.4 and later versions are installed.

Steps:

Step 1: PySpark - Read CSV File into DataFrame

1. **Initialize SparkSession:**

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
```

```
.appName("ReadCSV") \
.getOrCreate()
```

2. Read CSV File:

```
df = spark.read.csv("path/to/your/csvfile.csv", header=True, inferSchema=True)
```

3. Display DataFrame:

```
df.show()
```

4. Print Schema:

```
df.printSchema()
```

Step 2: Create and Query a Hive Table in PySpark

1. Enable Hive Support in SparkSession:

```
spark = SparkSession.builder \
    .appName("HiveExample") \
    .enableHiveSupport() \
    .getOrCreate()
```

2. Create Hive Database (if not exists):

```
spark.sql("CREATE DATABASE IF NOT EXISTS mydb")
spark.sql("USE mydb")
```

3. Create Hive Table:

```
spark.sql("""
CREATE TABLE IF NOT EXISTS my_table (
    id INT,
    name STRING,
    age INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
""")
```

4. Load Data into Hive Table:

```
spark.sql("LOAD DATA LOCAL INPATH 'path/to/your/csvfile.csv' INTO TABLE my_table")
```

5. Query Hive Table:

```
result = spark.sql("SELECT * FROM my_table")
result.show()
```

Conclusion:

In this lab assignment, we have learned how to perform two essential tasks using PySpark:

1. **Reading a CSV File into a DataFrame:**
 - You initialized a SparkSession and read a CSV file into a DataFrame.
 - You displayed the contents and schema of the DataFrame.
2. **Creating and Querying a Hive Table:**
 - You enabled Hive support in SparkSession and created a Hive database and table.
 - You loaded data from a CSV file into the Hive table and performed SQL queries on the table.

These skills are fundamental for working with large datasets in a distributed computing environment, allowing you to efficiently process and analyze data using the power of Apache Spark and Hive. By integrating these technologies, you can leverage the scalability of Spark and the SQL capabilities of Hive, making your data processing tasks more effective and streamlined.

Experiment no. 05

Aim: To implement the K-means clustering algorithm using R programming and analyze the results graphically. The analysis includes selecting an appropriate value for k , justifying the choice, and interpreting the output.

Theory:

K-means Clustering Algorithm:

K-means is a widely used unsupervised learning algorithm that partitions a dataset into k clusters. The goal is to minimize the variance within each cluster. The algorithm follows these steps:

1. **Initialization:** Choose k initial centroids randomly from the dataset.
2. **Assignment:** Assign each data point to the nearest centroid, forming k clusters.
3. **Update:** Calculate the new centroids by taking the mean of all data points in each cluster.
4. **Repeat:** Repeat the assignment and update steps until the centroids no longer change or a predefined number of iterations is reached.

Selecting the Value of k :

Choosing the right value for k is crucial for the performance of the K-means algorithm. Common methods to determine the optimal k include:

- **Elbow Method:** Plot the sum of squared distances from each point to its assigned centroid (within-cluster sum of squares, WCSS) for different values of k . The optimal k is typically at the "elbow" point, where the rate of decrease sharply slows.
- **Silhouette Analysis:** Measures how similar each point is to its own cluster compared to other clusters. The optimal k maximizes the average silhouette score.
- **Gap Statistic:** Compares the WCSS with that expected under a null reference distribution of the data.

Experiment Details

Implementation in R:

1. **Load the necessary libraries:**

```
library(ggplot2)
```

```
library(cluster)
```

2. **Load and preprocess the dataset:**

```
data <- read.csv('path_to_dataset.csv')  
data <- na.omit(data) # Handle missing values
```

3. **Determine the optimal value of k using the Elbow Method:**

```
set.seed(123)  
wcss <- vector()  
for (i in 1:10) {
```



```

kmeans_model <- kmeans(data, centers = i)
wcss[i] <- sum(kmeans_model$tot.withinss)
}
plot(1:10, wcss, type = 'b', main = 'Elbow Method', xlab = 'Number of clusters (k)', ylab =
'WCSS')

```

4. Apply the K-means algorithm with the selected k

```

optimal_k <- 3 # Assume 3 is the optimal k from the Elbow Method
kmeans_result <- kmeans(data, centers = optimal_k)
data$cluster <- as.factor(kmeans_result$cluster)

```

5. Visualize the results:

```

ggplot(data, aes(x = Feature1, y = Feature2, color = cluster)) +
  geom_point() +
  labs(title = 'K-means Clustering', x = 'Feature 1', y = 'Feature 2')

```

Conclusion

In this experiment, we successfully implemented the K-means clustering algorithm in R. The Elbow Method was used to determine the optimal value of k, which was found to be 3. The clustering results were visualized, showing distinct clusters. The choice of k is crucial as it directly affects the cluster formation and overall analysis. By using the Elbow Method, we ensured a balance between simplicity and accuracy, avoiding both underfitting and overfitting. Further analysis, such as silhouette scores or the gap statistic, could be used to validate the chosen k.

Experiment no. 05

Aim: To compute TF-IDF (Term Frequency-Inverse Document Frequency) values of words from different types of corpora using R programming. The analysis will include:

1. A corpus with unique values.
2. A corpus with similar documents.
3. A single word repeated multiple times in multiple documents.

Theory:

TF-IDF (Term Frequency-Inverse Document Frequency):

TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a corpus. It is commonly used in information retrieval and text mining. TF-IDF is the product of two statistics, term frequency (TF) and inverse document frequency (IDF).

- **Term Frequency (TF):** Measures how frequently a term appears in a document.

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Inverse Document Frequency (IDF):** Measures how important a term is within the entire corpus

$$IDF(t, D) = \log \left(\frac{\text{Total number of documents in the corpus}}{\text{Number of documents containing term } t} \right)$$

- **TF-IDF:** Combines both measures.

$$TF\text{-}IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Experiment Details

Implementation in R:

1. **Load the necessary libraries:**

```
library(tm)
```

```
library(tidytext)
```

```
library(dplyr)
```

2. **Create the corpora:**

```
# Corpus with unique values
```

```
corpus_unique <- Corpus(VectorSource(c("apple banana cherry", "dog elephant fish", "grape hat ink")))
```

```
# Corpus with similar documents
```

```
corpus_similar <- Corpus(VectorSource(c("apple apple banana", "apple banana cherry", "banana cherry apple")))
```

```
# Corpus with a single word repeated multiple times
```

```
corpus_repeated <- Corpus(VectorSource(c("apple apple apple", "apple apple apple", "apple apple apple")))
```

3. Preprocess the text:

```
preprocess_corpus <- function(corpus) { corpus <- tm_map(corpus, content_transformer(tolower)) corpus <-  
<- tm_map(corpus, removePunctuation) corpus <- tm_map(corpus, removeNumbers) corpus <-  
tm_map(corpus, removeWords, stopwords("english")) corpus <- tm_map(corpus, stripWhitespace)  
return(corpus) } corpus_unique <- preprocess_corpus(corpus_unique) corpus_similar <-  
preprocess_corpus(corpus_similar) corpus_repeated <- preprocess_corpus(corpus_repeated)
```

4. Create Document-Term Matrices and compute TF-IDF values:

```
dtm_unique <- DocumentTermMatrix(corpus_unique)  
dtm_similar <- DocumentTermMatrix(corpus_similar)  
dtm_repeated <- DocumentTermMatrix(corpus_repeated)
```

```
tfidf_unique <- weightTfIdf(dtm_unique)  
tfidf_similar <- weightTfIdf(dtm_similar)  
tfidf_repeated <- weightTfIdf(dtm_repeated)
```

5. Convert to data frame for better readability:

```
tfidf_to_df <- function(tfidf) {  
  as.data.frame(as.matrix(tfidf))  
}  
  
df_tfidf_unique <- tfidf_to_df(tfidf_unique)  
df_tfidf_similar <- tfidf_to_df(tfidf_similar)  
df_tfidf_repeated <- tfidf_to_df(tfidf_repeated)  
  
df_tfidf_unique  
df_tfidf_similar  
df_tfidf_repeated
```

Conclusion:

In this experiment, we successfully computed TF-IDF values for words from three different types of corpora using R programming:

1. **Corpus with unique values:** Each document had distinct words, leading to a uniform distribution of TF-IDF values.
2. **Corpus with similar documents:** Similar documents resulted in higher TF-IDF values for common words, emphasizing their importance within the corpus.
3. **Single word repeated multiple times:** The repeated word had a high term frequency but a lower inverse document frequency, leading to high TF values but lower TF-IDF values.

The TF-IDF metric effectively highlighted the importance of words relative to the corpus, showcasing its utility in various text mining applications. Further analysis could involve visualizing these TF-IDF values to gain deeper insights.

Experiment No: 7

Aim: Analytical representation of Linear Regression using Movie recommendation dataset

Theory:

Regression shows a line or curve that passes through all the data points on the target-predictor graph in such a way that the vertical distance between the data points and the regression line is minimum.

There are two types of linear regression.

Simple linear regression: uses only one independent variable

Multiple linear regression: uses two or more independent variables

Linear Regression is a commonly used type of predictive analysis. Linear Regression is a statistical approach for modeling the relationship between a dependent variable and a given set of independent variables. It is predicted that a straight line can be used to approximate the relationship. The goal of linear regression is to identify the line that minimizes the discrepancies between the observed data points and the line's anticipated values. In Machine Learning Linear regression is one of the easiest and most popular Machine Learning algorithms.

- It is a statistical method that is used for predictive analysis.

- Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
- Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable changes according to the value of the independent variable.

It is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables. One variable denoted x is regarded as an independent variable and the other one denoted y is regarded as a dependent variable. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

- The dependent variable, also known as the response or outcome variable is represented by the letter Y.
- The independent variable, often known as the predictor or explanatory variable, is denoted by the letter X.
- The intercept, or value of Y when X is zero, is represented by the β_0 .

The slope or change in Y resulting from a one-unit change in X is represented by the β_1 .

The error term or the unexplained variation in Y is represented by the ε

The figure below illustrates the linear regression model, where

the best-fit regression line is in blue

the intercept (β_0) and the slope (β_1) are shown in green

the error terms (ε) are represented by vertical red lines

From the scatter plot above, it can be seen that not all the data points fall exactly on the fitted regression line. Some of the points are above the blue curve and some are below it; overall, the residual errors (ε) have approximately mean zero.

The sum of the squares of the residual errors are called the Residual Sum of Squares or RSS.

The average variation of points around the fitted regression line is called the Residual Standard Error (RSE). This is one of the metrics used to evaluate the overall quality of the fitted regression model. The lower the RSE, the better it is.

Linear Regression Line

A linear line showing the relationship between the dependent and independent variables is called a regression line. A regression line can show two types of relationship:

Positive Linear Relationship: If the dependent variable increases on the Y-axis and the independent variable increases on the X-axis, then such a relationship is termed as a Positive linear relationship.

Negative Linear Relationship: If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.

Example:

For understanding the concept let's consider a salary dataset where it is given the value of the dependent variable (salary) for every independent variable (years experienced).

Defined for general purposes:

x as a feature vector, i.e $x = [x_1, x_2, \dots, x_n]$,

y as a response vector, i.e $y = [y_1, y_2, \dots, y_n]$

for n observations (in the example, n=10).

| Years experienced | Salary |
|-------------------|--------|
|-------------------|--------|

| | |
|-----|----------|
| 1.1 | 39343.00 |
|-----|----------|

| | |
|-----|----------|
| 1.3 | 46205.00 |
|-----|----------|

| | |
|-----|----------|
| 1.5 | 37731.00 |
|-----|----------|

| | |
|-----|----------|
| 2.0 | 43525.00 |
|-----|----------|

| | |
|-----|----------|
| 2.2 | 39891.00 |
|-----|----------|

| | |
|-----|----------|
| 2.9 | 56642.00 |
|-----|----------|

| | |
|-----|----------|
| 3.0 | 60150.00 |
|-----|----------|

| | |
|-----|----------|
| 3.2 | 54445.00 |
|-----|----------|

| | |
|-----|----------|
| 3.2 | 64445.00 |
|-----|----------|

| | |
|-----|----------|
| 3.7 | 57189.00 |
|-----|----------|

R CODE

```
# 1 Create the data frame
```

```
data <- data.frame(
```

```
  Years_Exp = c(1.1, 1.3, 1.5, 2.0, 2.2, 2.9, 3.0, 3.2, 3.2, 3.7),
```

```
  Salary = c(39343.00, 46205.00, 37731.00, 43525.00,
```

```
            39891.00, 56642.00, 60150.00, 54445.00, 64445.00, 57189.00))
```

2 Create the scatter plot

```
plot(data$Years_Exp, data$Salary,  
      xlab = "Years Experienced",  
      ylab = "Salary",  
      main = "Scatter Plot of Years Experienced vs Salary")
```

Now, let's find a line that fits the above scatter plot through which we can predict any value of y or response for any value of x

The line which best fits is called the Regression line.

The equation of the regression line is given by: $y = a + bx$

Where y is the predicted response value, a is the y-intercept,

x is the feature value and b is the slope.

To create the model, evaluate the values of regression coefficients a and b. And as soon as the estimation of these coefficients is done, the response model can be predicted using the Least Square Technique.

The basic syntax for regression analysis in R is

```
lm(Y ~ model)
```

3 implement Simple Linear Regression:

```
install.packages('caTools')*
```

```
library(caTools)
```

```
split = sample.split(data$Salary, SplitRatio = 0.7)
```

```
trainingset = subset(data, split == TRUE)
```

```
testset = subset(data, split == FALSE)
```

Fitting Simple Linear Regression to the Training set

```
lm.r= lm(formula = Salary ~ Years_Exp,
```

```
      data = trainingset)
```

#Summary of the model

```
summary(lm.r)
```

*The caTools package in R Programming Language is a versatile and widely used package that provides a collection of tools for data analysis, including functions for splitting data, running moving averages, and performing various mathematical and statistical operations.

Output:

Call: Using the “lm” function, we will be performing a regression analysis of “Salary” against “Years_Exp” according to the formula displayed on this line

Residuals: Each residual in the “Residuals” section denotes the difference between the actual salaries and predicted values. These values are unique to each observation in the data set. For instance, observation 1 has a residual of 463.1.

Coefficients: Linear regression coefficients are revealed within the contents of this section.

(Intercept): The estimated salary when Years_Exp is zero is 30927, which represents the intercept for this case.

Years_Exp: For every year of experience gained, the expected salary is estimated to increase by 7230 units according to the coefficient for “Years_Exp”. This coefficient value suggests that each year of experience has a significant impact on the estimated salary.

Estimate: The model’s estimated coefficients can be found in this column.

Std. Error: “More precise estimates” can be deduced from smaller standard errors that are a gauge of the ambiguity that comes along with coefficient estimates.

t value: The coefficient estimate’s standard error distance from zero is measured by the t-value. Its purpose is to examine the likelihood of the coefficient being zero by testing the null hypothesis. A higher t-value’s absolute value indicates a higher possibility of statistical significance pertaining to the coefficient.

Pr(>|t|): This column provides the p-value associated with the t-value. The p-value indicates the probability of observing the t-statistic (or more extreme) under the null hypothesis that the coefficient is zero. In this case, the p-value for the intercept is 0.00144, and for “Years_Exp,” it is 0.01482.

Signif. codes: These codes indicate the level of significance of the coefficients.

Residual standard error: This is a measure of the variability of the residuals. In this case, it’s 4944, which represents the typical difference between the actual salaries and the predicted salaries.

Multiple R-squared: R-squared (R^2) is a measure of the goodness of fit of the model. It represents the proportion of the variance in the dependent variable that is explained by the independent variable(s). In this case, the R-squared is 0.7266, which means that approximately 72.66% of the variation in salaries can be explained by years of experience.

Adjusted R-squared: The adjusted R-squared adjusts the R-squared value based on the number of predictors in the model. It accounts for the complexity of the model. In this case, the adjusted R-squared is 0.6719.

F-statistic: The F-statistic is used to test the overall significance of the model. In this case, the F-statistic is 13.29 with 1 and 5 degrees of freedom, and the associated p-value is 0.01482. This p-value suggests that the model as a whole is statistically significant.

In summary, this linear regression analysis suggests that there is a significant relationship between years of experience (Years_Exp) and salary (Salary). The model explains approximately 72.66% of the variance in salaries, and both the intercept and the coefficient for “Years_Exp” are statistically significant at the 0.01 and 0.05 significance levels, respectively.

Recommendation System

A recommendation system is an artificial intelligence or AI algorithm, usually associated with machine learning that uses Big Data to suggest or recommend additional products to consumers. These can be based on various criteria, including past purchases, search history, demographic information, and other factors. Recommender systems are highly useful as they help users discover products and services they might otherwise have not found on their own.

Types of Recommendation Systems:

- Collaborative filtering algorithms recommend items (this is the filtering part) based on preference information from many users (this is the collaborative part).
- Content filtering, by contrast, uses the attributes or features of an item (this is the content part) to recommend other items similar to the user’s preferences.
- Hybrid recommender systems combine the advantages of the types above to create a more comprehensive recommending system.
- Context filtering includes users’ contextual information in the recommendation process.

Assignment:

- 1.Linear Regression Analysis on House Price.
- 2.Using the Simple Linear Regression predict the Happiness rate based on the Income.
- 3.Consider to evaluate the impact of advertising budgets of three Medias (youtube, facebook and newspaper) on future sales. Is this example of a problem that can be modeled with linear regression?
- 4.Advantages and Drawbacks of the Liner Regression model.
- 5.Difference between Liner and Nonlinear regression models.

CASE STUDY:

how Netflix recommendation engine works

Conclusion:

Experiment No.8

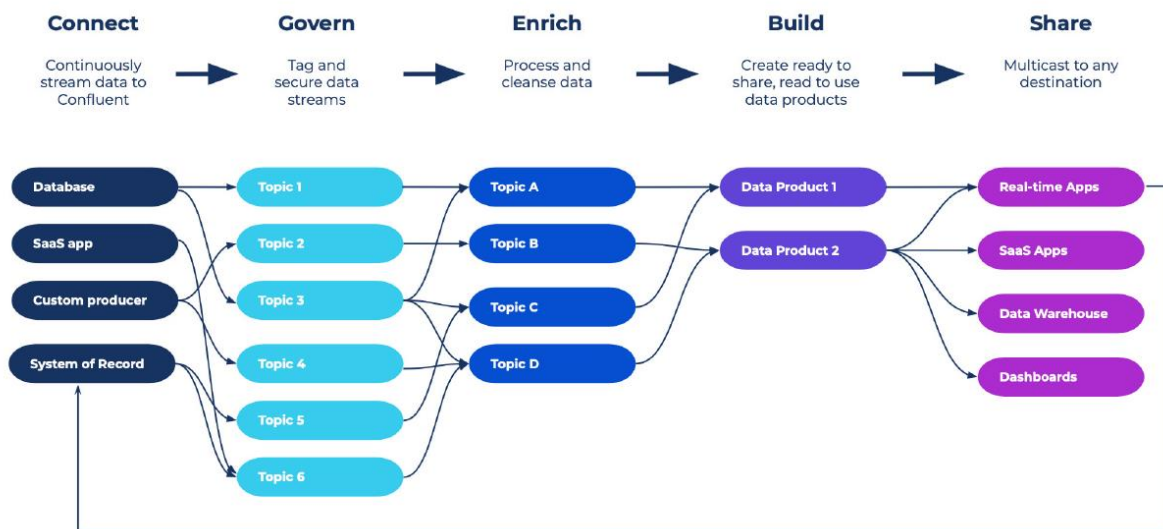
AIM: Data streaming and pipelining using online cloud platform using confluent for Kafka cloud Theory

To make data more accessible, most IT organizations centralize as much information as possible. They typically use point-to-point data pipelines to move data between operational databases and a centralized data warehouse or lake. For example, extract, transform, and load (ETL) pipelines ingest data, transform it in periodic batches, and later push it downstream to an analytical data warehouse. ETL pipelines—and reverse ETL pipelines—can also share the results of data analysis that takes place in the warehouse back to operational databases and applications.

Getting started with streaming data pipelines

Streaming data pipelines are a modern approach to delivering data as a self-service product. Instead of sending data to a centralized warehouse or an analytics tool before making it available to applications, streaming data pipelines can capture changes to data in real time, enrich them on the fly, and send them to downstream systems. And because teams get self-service access, they can find, browse, create, share, and reuse data—wherever and whenever it's needed.

Confluent makes it easy and cost-effective to build streaming data pipelines and to evolve them as business and data needs change. Using Confluent, you can build and deploy modern data flows in five simple steps as shown in figure:



1. Connect

Create and manage data flows with an easy-to-use visual user interface and pre-built connectors

2 Govern

Centrally manage, tag, audit, and apply policies for trusted high-quality data streams

3 Enrich

Use SQL to combine, aggregate, clean, process, and shape data in real time, increasing the safety, efficiency, and usability of your data streams to power operational, analytical, and business intelligence use cases.

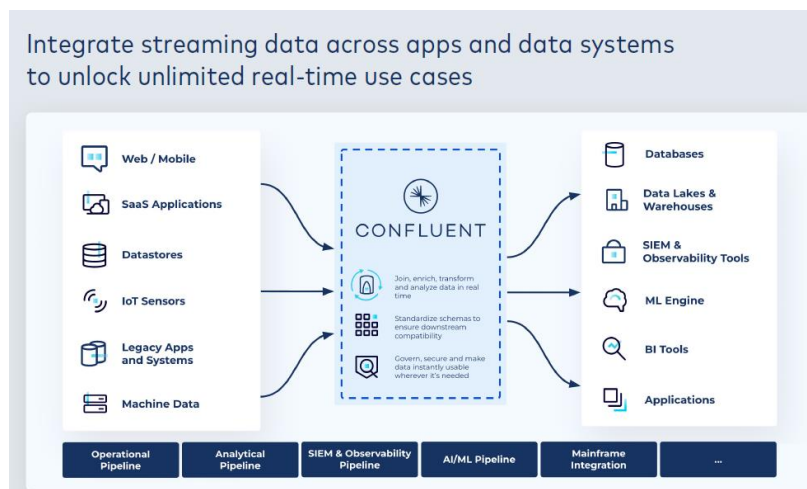
4 **Build** Prepare well-formatted, trustworthy data products for downstream systems and apps.

5 **Share** Collaborate securely on live streams with self-service data discovery and sharing.

Streaming data pipelines will help to solve enterprise data needs today, and it will serve the business needs in the future.

With streaming data pipelines from Confluent, it will transform your business and meet ambitious goals such as these:

- Power all use cases with high-quality, real-time data streams
- Bring new products to market faster with self-service data access
- Boost developer productivity by simplifying data flow development and iteration
- Enable agile pipeline development to meet changing data needs
- Accelerate innovation while maintaining trust and governance



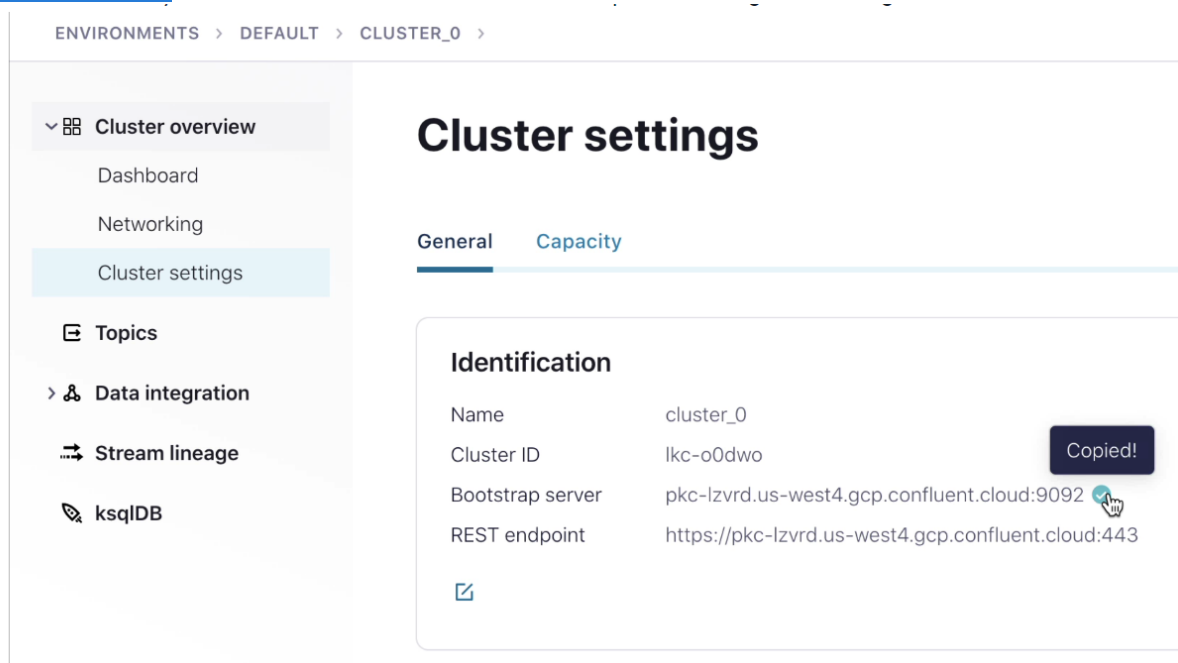
Implementation steps:

Kafka Setup

We are going to need a Kafka Cluster for our client application to operate with. This dialog can help you configure your Confluent Cloud cluster, create a Kafka cluster for you, or help you input an existing cluster bootstrap server to connect to.

First, sign up for a free [Confluent Cloud](#) account if you don't already have one. You will get \$400 in credits when you sign up. To avoid having to enter a credit card, navigate to [Billing & payment](#), scroll to the bottom, and add the promo code **CONFLUENTDEV1**. With this promo code, you will not have to enter your credit card info for 30 days or until your credits run out.

After you login to the Confluent Cloud Console and provision your Kafka cluster, note your Confluent Cloud bootstrap server as we will need it to configure the producer and consumer clients in upcoming steps. You can obtain your Confluent Cloud Kafka cluster bootstrap server configuration using the [Confluent Cloud Console](#):



Next, choose the authentication mechanism that the producer and consumer client applications will use to access Confluent Cloud: either [basic authentication](#) or [OAuth](#).

Basic authentication is quicker to implement since you only need to create an API key in Confluent Cloud, whereas OAuth requires that you have an OAuth provider, as well as an OAuth application created within it for use with Confluent Cloud, in order to proceed.

Select your authentication mechanism: Basic

You can use the [Confluent Cloud Console](#) to create a key for you by navigating to the **API Keys** section under **Cluster Overview**.

CONFLUENT

Stream Catalog LEARN

HOME > ENVIRONMENTS > DEFAULT > TEST-CLUSTER > API KEYS >

Cluster Overview

Dashboard

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

Schema Registry

Create key

1. Access control

Select scope for API key

Global access

Allow your API key to access everything you can access. Key access will be linked to your account.

**Recommended for development.*

Granular access

Limit the access for your API key. Manage your API key's access through a service account.

**Recommended for production.*

Next Cancel

Note the API key and secret as we will use them when configuring the producer and consumer clients in upcoming steps.

Create Topic

A topic is an immutable, append-only log of events. Usually, a topic is comprised of the same kind of events, e.g., in this guide we create a topic for retail purchases.

Create a new topic, **purchases**, which you will use to produce and consume events.

CONFLUENT

Stream Catalog LEARN

HOME > ENVIRONMENTS > DEFAULT > TEST-CLUSTER > TOPICS >

Cluster Overview

Dashboard

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

New topic

Topic name* purchases

Partitions* 1

[Show advanced settings](#)

Cancel

Create with defaults

When using Confluent Cloud, you can use the [Confluent Cloud Console](#) to create a topic. Create a topic with 1 partition and defaults for the remaining settings.

Build

Producer

Let's create the Python producer application by pasting the following code into a file **producer.py**.

Fill in the appropriate **bootstrap.servers** value and any additional security configuration needed inline where the client configuration **config** object is created.

Build Consumer

Next, create the Python consumer application by pasting the following code into a file `consumer.py`.

Again, fill in the appropriate `bootstrap.servers` value and any additional security configuration needed inline where the client configuration `config` object is created.

Produce Events

Make the producer script executable, and run it:

Result:

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 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 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.

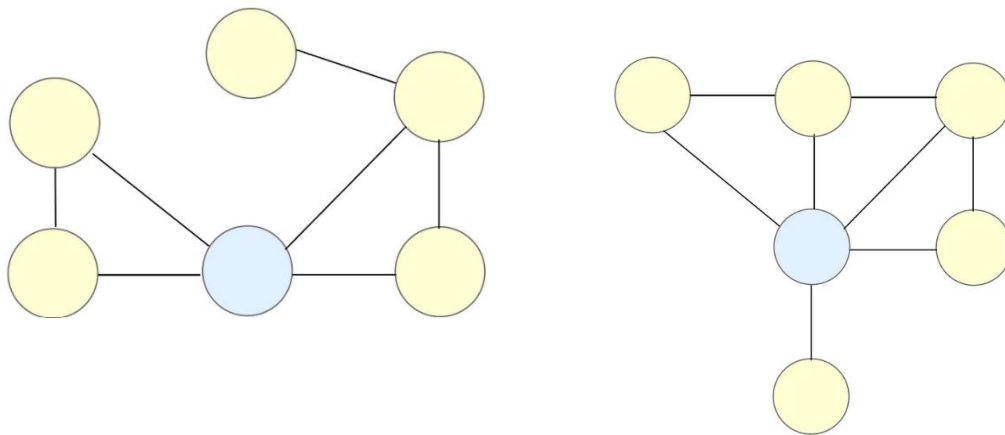
The graph on the left is more dense and this could be due to the fact that Yelpers that have been on the platform longer have had more time to build their reputation on Yelp and establish themselves in communities. On the other hand, Yelpers that joined in 2015 have less dense communities, less edges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

From our subgraphs of communities, we can detect **cliques**:

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.

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.

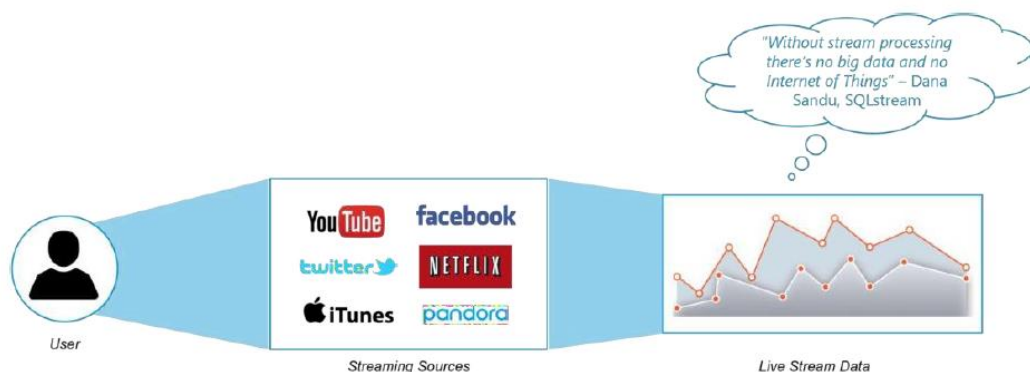


Bridges are important because they connect two different groups in a social network; they are useful in bridging the gap between different communities! An example of a bridge is someone that is able to communicate and interpret data to both tech and non-tech team members. Another example is when your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

Conclusion:

Experiment 10

Aim: Write a program to implement Twitter Sentiment Analysis System where we populate real-time sentiments for crisis management, service adjusting and target marketing using PYSpark

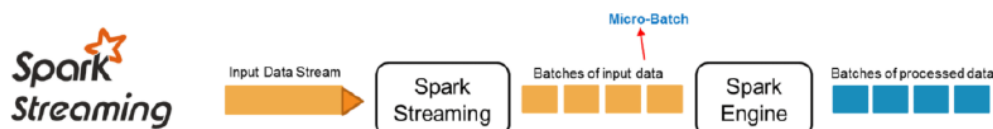


Why Spark Streaming?

We can use Spark Streaming to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.

Spark Streaming Overview

Spark Streaming is used for processing real-time streaming data. It is a useful addition to the core Spark API. Spark Streaming enables high-throughput and fault-tolerant stream processing of live data streams.



Spark Streaming Features

1. **Scaling:** Spark Streaming can easily scale to hundreds of nodes.

2. **Speed:** It achieves low latency.
3. **Fault Tolerance:** Spark has the ability to efficiently recover from failures.
4. **Integration:** Spark integrates with batch and real-time processing.
5. **Business Analysis:** Spark Streaming is used to track the behavior of customers which can be used in business analysis.

Spark Streaming Fundamentals

1. Streaming Context
2. DStream
3. Caching
4. Accumulators, Broadcast Variables and Checkpoints

Streaming Context

Streaming Context consumes a stream of data in Spark. It registers an *Input DStream* to produce a *Receiver* object. It is the main entry point for Spark functionality. Spark provides a number of default implementations of sources like Twitter, Akka Actor and ZeroMQ that are accessible from the context.

DStream

Discretized Stream (DStream) is the basic abstraction provided by Spark Streaming. It is a continuous stream of data. It is received from a data source or a processed data stream generated by transforming the input stream.

Caching

DStreams allow developers to cache/ persist the stream's data in memory. This is useful if the data in the DStream will be computed multiple times. This can be done using the *persist()* method on a DStream

Accumulators, Broadcast Variables and Checkpoints

Accumulators: *Accumulators* are variables that are only added through an associative and commutative operation. They are used to implement counters or sums. Tracking

Conclusion: