**NAME: Vedashree Bhalerao CLASS: AIA-1 ROLL NO:2213191 BATCH: B**

# Assignment No 1

**Title:** Getting data to work with R Programming:

i. 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.

1. Measuring the length of the data set. Count the number of lines in the file.
2. Encode the categorical data vi. Plot a graph and give your insights for the application selected cases.

**Theory:**

## 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:

# 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:

# 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:

# 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:

# 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.

# 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:

# 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:

# 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.

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.

# 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.

# 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:

# 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:

# 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.
* **Link for dataset** [**https://www.kaggle.com/datasets/tomaslui/healthcaredataset?resource=download**](https://www.kaggle.com/datasets/tomaslui/healthcare-dataset?resource=download)

**Conclusion:** We have understood the environment of R Programming and basic commands of R programming. We have also used the concepts of data frames and operations on it. Further we have visualised the data using different graphs.

**CODE:**

install.packages("ggplot2") install.packages("dplyr") install.packages("tidyr") library(ggplot2) library(dplyr) library(tidyr) data<- read.csv("Iris.csv") summary(data) barplot(data$sepal\_length, col = "green") boxplot(data[,0:4]) plot(data$sepal\_length,data$sepal\_width, pch=18) hist(data$sepal\_length, col="purple",border="black")

pairs(~data$sepal\_length+data$sepal\_width+data$petal\_length+data$petal\_w idth)

# Install and load necessary packages install.packages("plotly") library(plotly) set.seed(123) data <- data.frame(

1. = rnorm(100, mean = 20, sd = 5),
2. = rnorm(100, mean = 30, sd = 8),
3. = rnorm(100, mean = 40, sd = 10)

)

plot <- plot\_ly(data, x = ~X, y = ~Y, z = ~Z, type = "scatter3d", mode =

"markers", marker = list(size = 5, color = "blue")) %>% layout(scene = list( xaxis = list(title = "X Variable"), yaxis = list(title = "Y Variable"), zaxis = list(title = "Z Variable")

))

print(plot)

**OUTPUT:**











