Biochemical Oxygen Demand

29-Oct-24

Name: Faizan

Registration # FA21-BSE-011

Dataset Assigned

BOD (Biochemical Oxygen Demand).

GitHub Link of Implementation

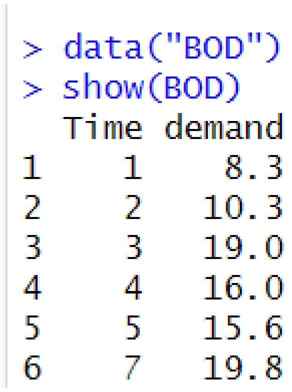
https://github.com/Faizan-Kh/Data-Science

Minkowski Distance

Minkowski distance is the generalization of Euclidean distance. Euclidean Distance Length of the line segment between two points in a Euclidean space is known as the Euclidean Distance [1].

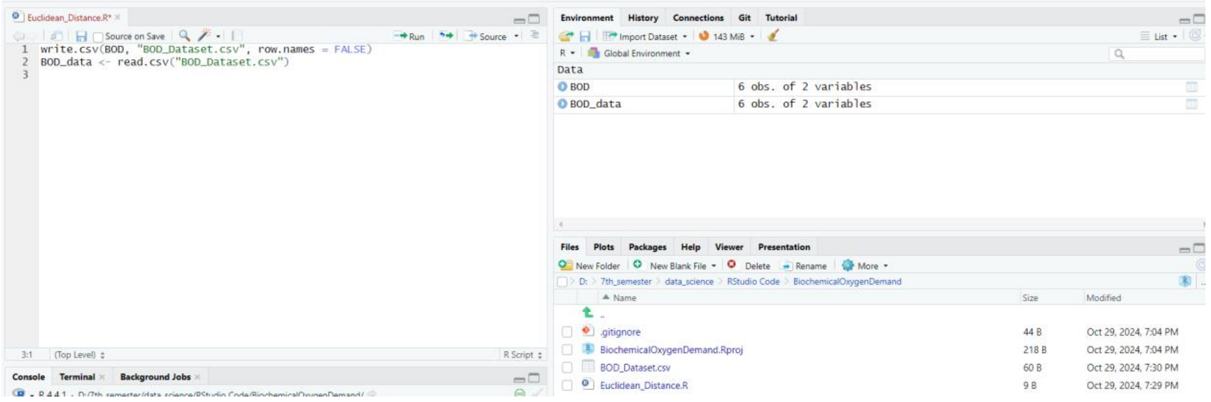
RStudio implementation

1) Loading the dataset to memory:



```
> data("BOD")
> show(BOD)
  Time demand
1    1    8.3
2    2   10.3
3    3   19.0
4    4   16.0
5    5   15.6
6    7   19.8
```

2) Exporting the dataset to a CSV file and importing it



```
write.csv(BOD, "BOD_Dataset.csv", row.names = FALSE)
BOD_data <- read.csv("BOD_Dataset.csv")
```

```
Data
BOD         6 obs. of 2 variables
BOD_data    6 obs. of 2 variables
```

D: > 7th_semester > data_science > RStudio Code > BiochemicalOxygenDemand

Name	Size	Modified
.gitignore	44 B	Oct 29, 2024, 7:04 PM
BiochemicalOxygenDemand.Rproj	218 B	Oct 29, 2024, 7:04 PM
BOD_Dataset.csv	60 B	Oct 29, 2024, 7:30 PM
Euclidean_Distance.R	9 B	Oct 29, 2024, 7:29 PM

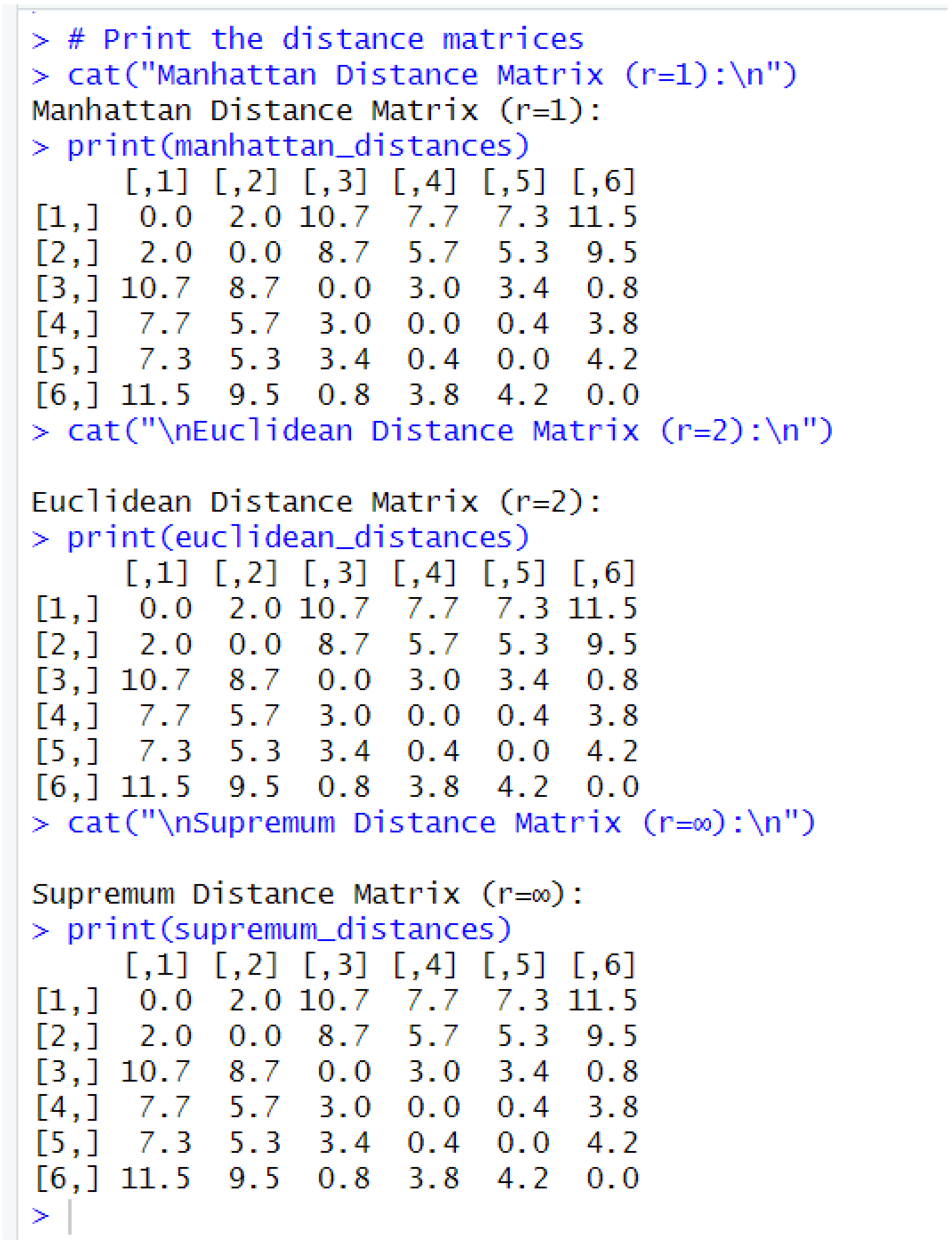
3) Calculating the Euclidean, city block and supremum distance by putting r= 1, 2, and ∞ on the imported dataset

```
# Load the BOD dataset and save it to a CSV

write.csv(BOD, "BOD_Dataset.csv", row.names = FALSE)
```

```r
# Read the dataset back into R
BOD_data <- read.csv("BOD_Dataset.csv")

# Define a generalized Minkowski distance function
minkowski_dist <- function(x, y, r) {
  if (r == Inf) {
    return(max(abs(x - y)))  # L∞ norm (Supremum distance)
  } else {
    return((sum(abs(x - y)^r))^(1/r))  # Lr norm (Minkowski
distance)
  }
}

# Get the number of demands
n <- length(BOD_data$demand)

# Initialize matrices to hold the distances
manhattan_distances <- matrix(NA, n, n)
euclidean_distances <- matrix(NA, n, n)
supremum_distances <- matrix(NA, n, n)

# Set values for r
r_values <- c(1, 2, Inf)  # 1 = Manhattan, 2 = Euclidean, Inf =
Supremum

# Calculate the distances for each pair
for (i in 1:n) {
  for (j in 1:n) {
    manhattan_distances[i, j] <-
minkowski_dist(BOD_data$demand[i], BOD_data$demand[j],
r_values[1])
```

```
    euclidean_distances[i, j] <-
minkowski_dist(BOD_data$demand[i], BOD_data$demand[j],
r_values[2])

    supremum_distances[i, j] <-
minkowski_dist(BOD_data$demand[i], BOD_data$demand[j],
r_values[3])

  }

}


# Print the distance matrices

cat("Manhattan Distance Matrix (r=1):\n")

print(manhattan_distances)

cat("\nEuclidean Distance Matrix (r=2):\n")

print(euclidean_distances)

cat("\nSupremum Distance Matrix (r=∞):\n")

print(supremum_distances)

print(distances)
```

Output



```
> # Print the distance matrices
> cat("Manhattan Distance Matrix (r=1):\n")
Manhattan Distance Matrix (r=1):
> print(manhattan_distances)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  0.0  2.0 10.7  7.7  7.3 11.5
[2,]  2.0  0.0  8.7  5.7  5.3  9.5
[3,] 10.7  8.7  0.0  3.0  3.4  0.8
[4,]  7.7  5.7  3.0  0.0  0.4  3.8
[5,]  7.3  5.3  3.4  0.4  0.0  4.2
[6,] 11.5  9.5  0.8  3.8  4.2  0.0
> cat("\nEuclidean Distance Matrix (r=2):\n")

Euclidean Distance Matrix (r=2):
> print(euclidean_distances)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  0.0  2.0 10.7  7.7  7.3 11.5
[2,]  2.0  0.0  8.7  5.7  5.3  9.5
[3,] 10.7  8.7  0.0  3.0  3.4  0.8
[4,]  7.7  5.7  3.0  0.0  0.4  3.8
[5,]  7.3  5.3  3.4  0.4  0.0  4.2
[6,] 11.5  9.5  0.8  3.8  4.2  0.0
> cat("\nSupremum Distance Matrix (r=∞):\n")

Supremum Distance Matrix (r=∞):
> print(supremum_distances)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  0.0  2.0 10.7  7.7  7.3 11.5
[2,]  2.0  0.0  8.7  5.7  5.3  9.5
[3,] 10.7  8.7  0.0  3.0  3.4  0.8
[4,]  7.7  5.7  3.0  0.0  0.4  3.8
[5,]  7.3  5.3  3.4  0.4  0.0  4.2
[6,] 11.5  9.5  0.8  3.8  4.2  0.0
> 
```

4)

SMC (Simple Matching Coefficient)

Also known as the statistic used for comparing the similarities between sample sets [2].

Implementation

1) We initialize a threshold to convert the data into binary

```
threshold <- mean(BOD_data$demand)

#Converting the data to binary
binary_demand <- ifelse(BOD_data$demand > threshold, 1, 0)
```

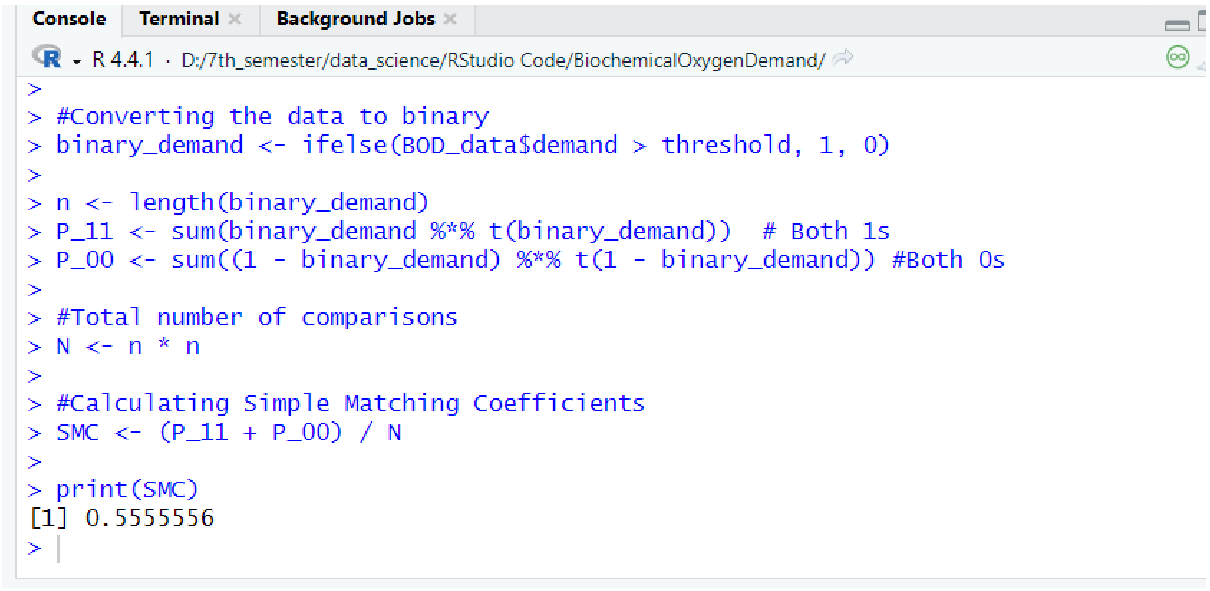
2) Treating both 1s and 0s

```
n <- length(binary_demand)
P_11 <- sum(binary_demand %*% t(binary_demand))  # Both 1s
P_00 <- sum((1 - binary_demand) %*% t(1 - binary_demand)) #Both 0s
```

3) Calculating the total number of similarities and SMC

```
#Total number of comparisons
N <- n * n

#Calculating Simple Matching Coefficients
SMC <- (P_11 + P_00) / N

print(SMC)
```

Output


Console
Terminal
Background Jobs
R 4.4.1 · D:/7th_semester/data_science/RStudio Code/BiochemicalOxygenDemand/

```
>
> #Converting the data to binary
> binary_demand <- ifelse(BOD_data$demand > threshold, 1, 0)
>
> n <- length(binary_demand)
> P_11 <- sum(binary_demand %*% t(binary_demand))  # Both 1s
> P_00 <- sum((1 - binary_demand) %*% t(1 - binary_demand)) #Both 0s
>
> #Total number of comparisons
> N <- n * n
>
> #Calculating Simple Matching Coefficients
> SMC <- (P_11 + P_00) / N
>
> print(SMC)
[1] 0.5555556
>
```

Jaccard Coefficient

Also known as Jaccard index, is a statistic used to find out the similarity and diversity of sample sets [3].

Implementation

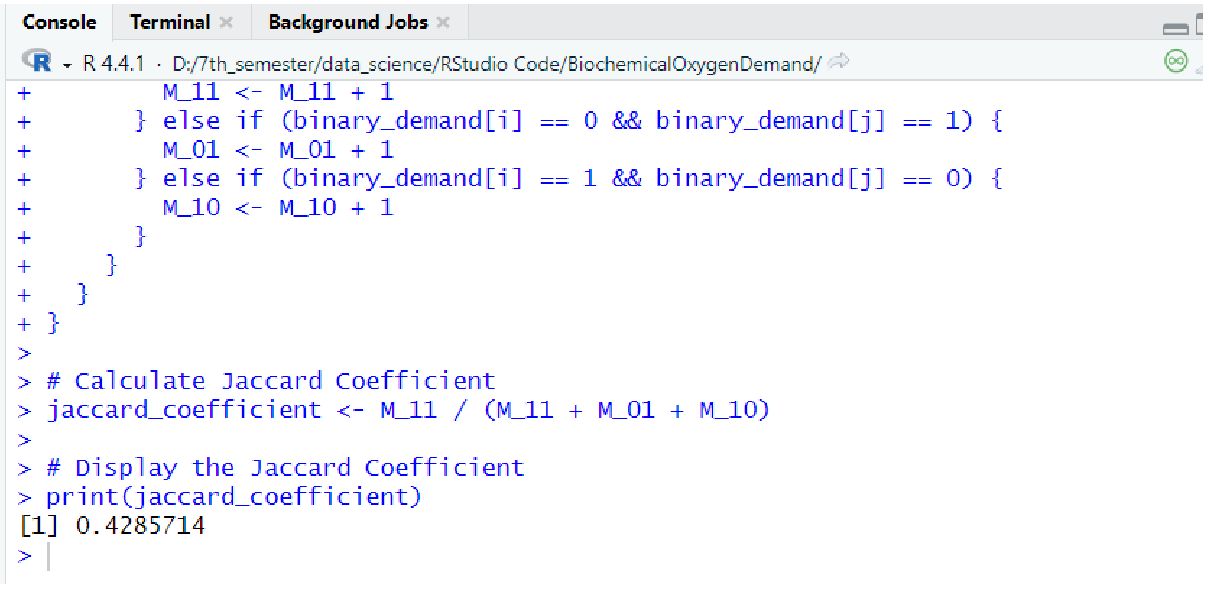
```
threshold <- 15  # Example threshold

binary_demand <- ifelse(BOD$demand > threshold, 1, 0)



n <- length(binary_demand)


# Initialize counts for M11, M01, and M10

M_11 <- 0

M_01 <- 0

M_10 <- 0


# Loop through each pair to count M11, M01, and M10

for (i in 1:n) {

  for (j in 1:n) {

    if (i != j) {

      if (binary_demand[i] == 1 && binary_demand[j] == 1) {

        M_11 <- M_11 + 1

      } else if (binary_demand[i] == 0 && binary_demand[j] == 1) {

        M_01 <- M_01 + 1

      } else if (binary_demand[i] == 1 && binary_demand[j] == 0) {

        M_10 <- M_10 + 1

      }

    }

  }

}
```

Calculate Jaccard Coefficient

jaccard_coefficient <- M_11 / (M_11 + M_01 + M_10)

print(jaccard_coefficient)

Output


Console
Terminal
Background Jobs
R 4.4.1 · D:/7th_semester/data_science/RStudio Code/BiochemicalOxygenDemand/

```
+           M_11 <- M_11 + 1
+         } else if (binary_demand[i] == 0 && binary_demand[j] == 1) {
+           M_01 <- M_01 + 1
+         } else if (binary_demand[i] == 1 && binary_demand[j] == 0) {
+           M_10 <- M_10 + 1
+         }
+       }
+     }
+ }
>
> # Calculate Jaccard Coefficient
> jaccard_coefficient <- M_11 / (M_11 + M_01 + M_10)
>
> # Display the Jaccard Coefficient
> print(jaccard_coefficient)
[1] 0.4285714
>
```

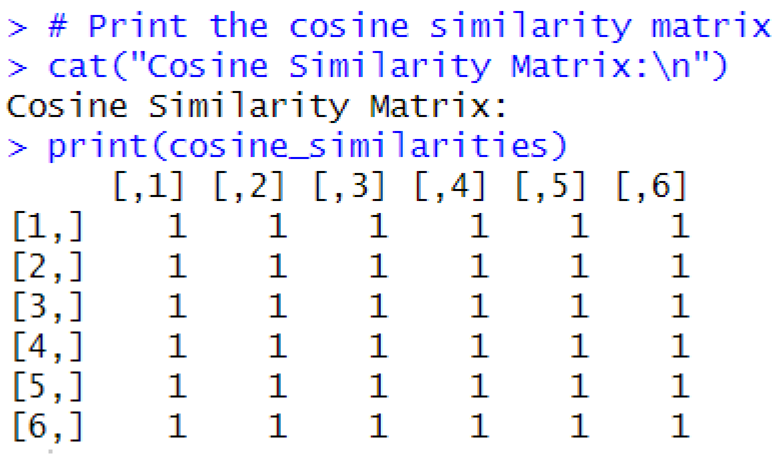
Cosine Similarity

The cosine measure computes the *angle* between the two documents, which is **insensitive** to the absolute length of the document

Implementation

```
# Load the BOD dataset and save it to a CSV

write.csv(BOD, "BOD_Dataset.csv", row.names = FALSE)


#or load it instead

#load("BOD")


# Read the dataset back into R

BOD_data <- read.csv("BOD_Dataset.csv")


# Define a function to calculate cosine similarity

cosine_similarity <- function(x, y) {
  dot_product <- sum(x * y)
  magnitude_x <- sqrt(sum(x^2))
  magnitude_y <- sqrt(sum(y^2))

  # Calculate cosine similarity
  if (magnitude_x == 0 || magnitude_y == 0) {
    return(NA)  # Avoid division by zero
  } else {
    return(dot_product / (magnitude_x * magnitude_y))
  }
}

# Get the number of demands
n <- length(BOD_data$demand)

# Initialize a matrix to hold the cosine similarities
cosine_similarities <- matrix(NA, n, n)

# Calculate the cosine similarity for each pair
for (i in 1:n) {
  for (j in 1:n) {
    cosine_similarities[i, j] <- 
cosine_similarity(BOD_data$demand[i], BOD_data$demand[j])
  }
}

# Print the cosine similarity matrix
cat("Cosine Similarity Matrix:\n")
print(cosine_similarities)
```

Output



```
> # Print the cosine similarity matrix
> cat("Cosine Similarity Matrix:\n")
Cosine Similarity Matrix:
> print(cosine_similarities)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    1    1    1    1
[2,]    1    1    1    1    1    1
[3,]    1    1    1    1    1    1
[4,]    1    1    1    1    1    1
[5,]    1    1    1    1    1    1
[6,]    1    1    1    1    1    1
```

Correlation

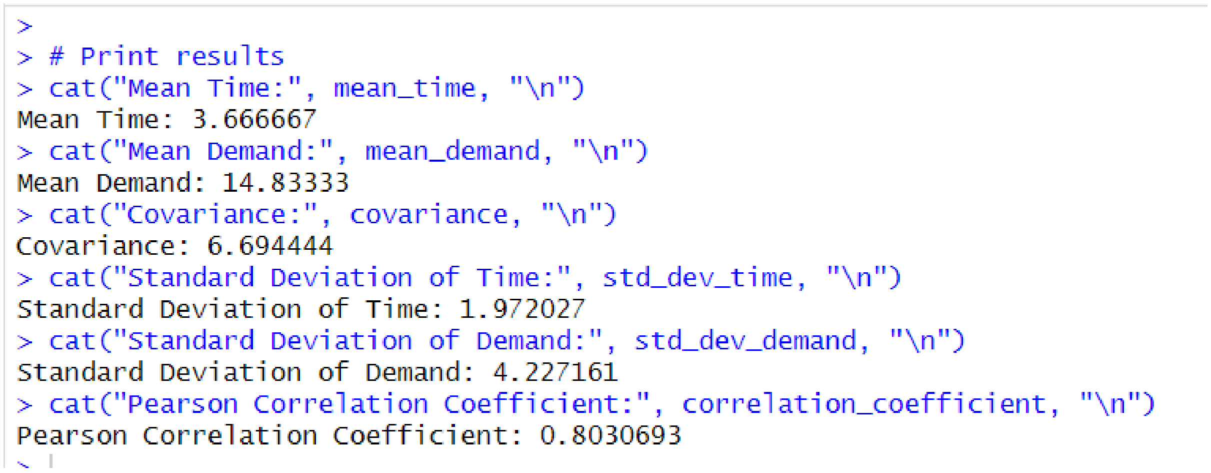
- Correlation measures the **linear relationship** between the attributes of the objects $y_k=ax_k+b$ for binary or continuous variable
- Measure of linear dependency b/w two variables x and y

Implementation

```
# Load the BOD dataset

BOD_data <- read.csv("BOD_Dataset.csv")


# Extract the Time and Demand columns

time <- BOD_data$Time

demand <- BOD_data$demand


# Calculate means

mean_time <- mean(time)

mean_demand <- mean(demand)


# Calculate covariance

covariance <- sum((time - mean_time) * (demand - mean_demand)) /
length(time)


# Calculate standard deviations

std_dev_time <- sqrt(sum((time - mean_time)^2) / length(time))
```

```
std_dev_demand <- sqrt(sum((demand - mean_demand)^2) /
length(demand))


# Calculate Pearson correlation coefficient

correlation_coefficient <- covariance / (std_dev_time *
std_dev_demand)


# Print results

cat("Mean Time:", mean_time, "\n")

cat("Mean Demand:", mean_demand, "\n")

cat("Covariance:", covariance, "\n")

cat("Standard Deviation of Time:", std_dev_time, "\n")

cat("Standard Deviation of Demand:", std_dev_demand, "\n")

cat("Pearson Correlation Coefficient:", correlation_coefficient,
"\n")
```

Output



```
>
> # Print results
> cat("Mean Time:", mean_time, "\n")
Mean Time: 3.666667
> cat("Mean Demand:", mean_demand, "\n")
Mean Demand: 14.83333
> cat("Covariance:", covariance, "\n")
Covariance: 6.694444
> cat("Standard Deviation of Time:", std_dev_time, "\n")
Standard Deviation of Time: 1.972027
> cat("Standard Deviation of Demand:", std_dev_demand, "\n")
Standard Deviation of Demand: 4.227161
> cat("Pearson Correlation Coefficient:", correlation_coefficient, "\n")
Pearson Correlation Coefficient: 0.8030693
```

Clustering using the distance matrix

# Load the BOD dataset and save it to a CSV

write.csv(BOD, "BOD_Dataset.csv", row.names = FALSE)

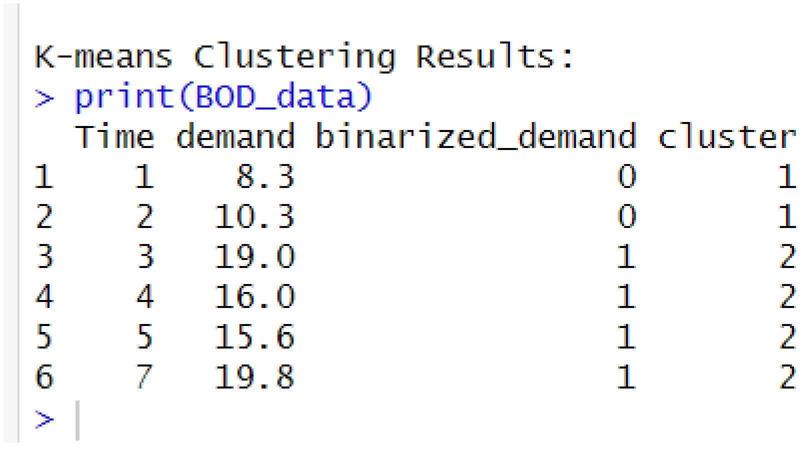
```
# Read the dataset back into R

BOD_data <- read.csv("BOD_Dataset.csv")
```

```
# Binarize the demand data based on a threshold
threshold <- 15
BOD_data$binarized_demand <- ifelse(BOD_data$demand > threshold,
1, 0)

# Define the distance functions
euclidean_dist <- function(x, y) {
  sqrt(sum((x - y) ^ 2))
}

minkowski_dist <- function(x, y, r) {
  sum(abs(x - y) ^ r)^(1/r)
}

cosine_similarity <- function(x, y) {
  dot_product <- sum(x * y)
  magnitude_x <- sqrt(sum(x^2))
  magnitude_y <- sqrt(sum(y^2))

  if (magnitude_x == 0 || magnitude_y == 0) {
    return(NA)  # Avoid division by zero
  } else {
    return(dot_product / (magnitude_x * magnitude_y))
  }
}

# Get the number of demands
n <- nrow(BOD_data)
demand_matrix <- as.matrix(BOD_data$demand)

# Calculate distance matrices
```

```
euclidean_matrix <- matrix(NA, n, n)
minkowski_matrix <- matrix(NA, n, n)
cosine_matrix <- matrix(NA, n, n)

# Calculate the distance matrices
for (i in 1:n) {
  for (j in 1:n) {
    # Using demand values for distance measures
    euclidean_matrix[i, j] <- euclidean_dist(demand_matrix[i, , 
drop = FALSE], demand_matrix[j, , drop = FALSE])
    minkowski_matrix[i, j] <- minkowski_dist(demand_matrix[i, , 
drop = FALSE], demand_matrix[j, , drop = FALSE], r = 3) # r can 
be 1, 2, or 3
    cosine_matrix[i, j] <- cosine_similarity(demand_matrix[i, , 
drop = FALSE], demand_matrix[j, , drop = FALSE])
  }
}

# Print the distance matrices
cat("Euclidean Distance Matrix:\n")
print(euclidean_matrix)
cat("\nMinkowski Distance Matrix:\n")
print(minkowski_matrix)
cat("\nCosine Similarity Matrix:\n")
print(cosine_matrix)

# Clustering using K-means based on Euclidean distance
set.seed(123) # For reproducibility
k <- 2 # Number of clusters
# Since K-means uses Euclidean distance, we will use the 
original demand matrix
kmeans_result <- kmeans(demand_matrix, centers = k)
```

```
# Add cluster assignment to the original dataset
BOD_data$cluster <- kmeans_result$cluster

# Print the clustering result
cat("\nK-means Clustering Results:\n")
print(BOD_data)
```

Output



```
K-means Clustering Results:
> print(BOD_data)
  Time demand binarized_demand cluster
1    1    8.3                0       1
2    2   10.3                0       1
3    3   19.0                1       2
4    4   16.0                1       2
5    5   15.6                1       2
6    7   19.8                1       2
> 
```

References

[1]. https://en.wikipedia.org/wiki/Euclidean_distance
[2]. https://en.wikipedia.org/wiki/Simple_matching_coefficient#:~:text=The%20simple%20matching%20coefficient%20(SMC,and%20diversity%20of%20sample%20sets.&text=value%201%2C%20and-,is%20the%20total%20number%20of%20attributes%20where%20A%20has,and%20B%20has%20value%200.
[3]. https://en.wikipedia.org/wiki/Jaccard_index