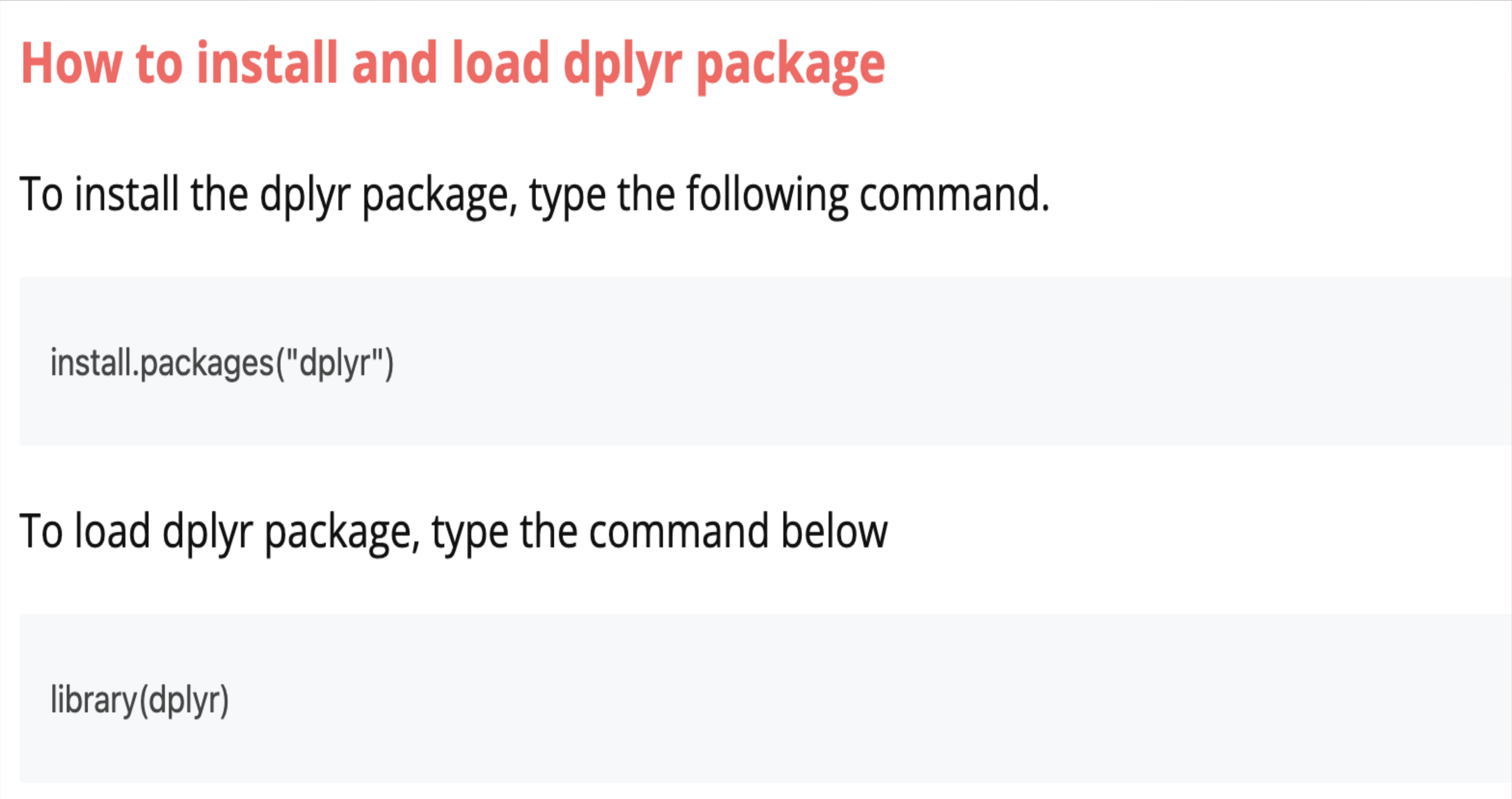
MODULE-4

DATA CLEANING AND SUMMARIZING WITH DPLYR PACKAGE

WHAT IS DPLYR?

- The dplyr is a powerful R-package to manipulate, clean and summarize unstructured data. In short, it makes data exploration and data manipulation easy and fast in R.

- The package "dplyr" comprises many functions that perform mostly used data manipulation operations such as applying filter, selecting specific columns, sorting data, adding or deleting columns and aggregating data.



How to install and load dplyr package

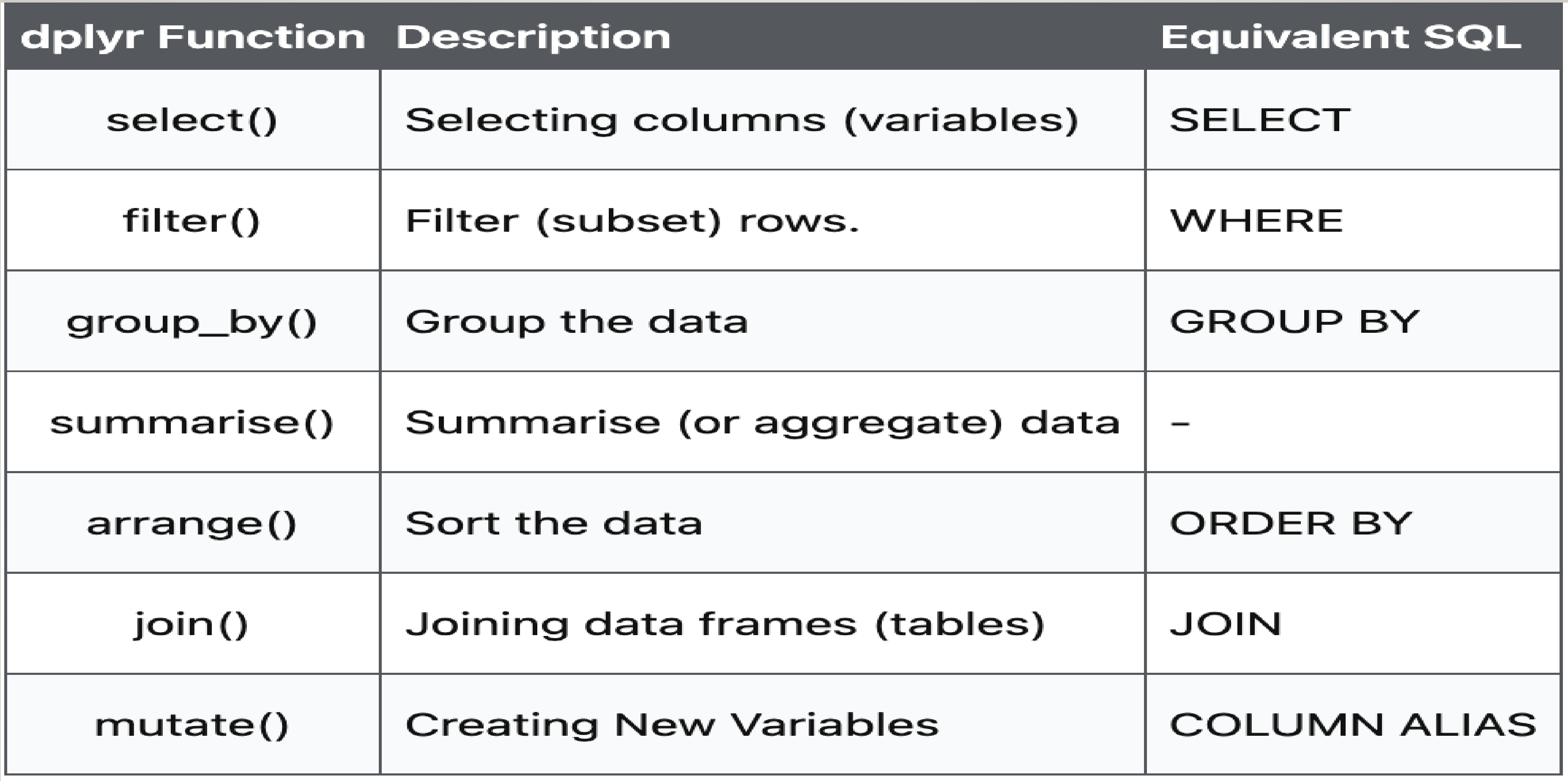
To install the dplyr package, type the following command.

```
install.packages("dplyr")
```

To load dplyr package, type the command below

```
library(dplyr)
```

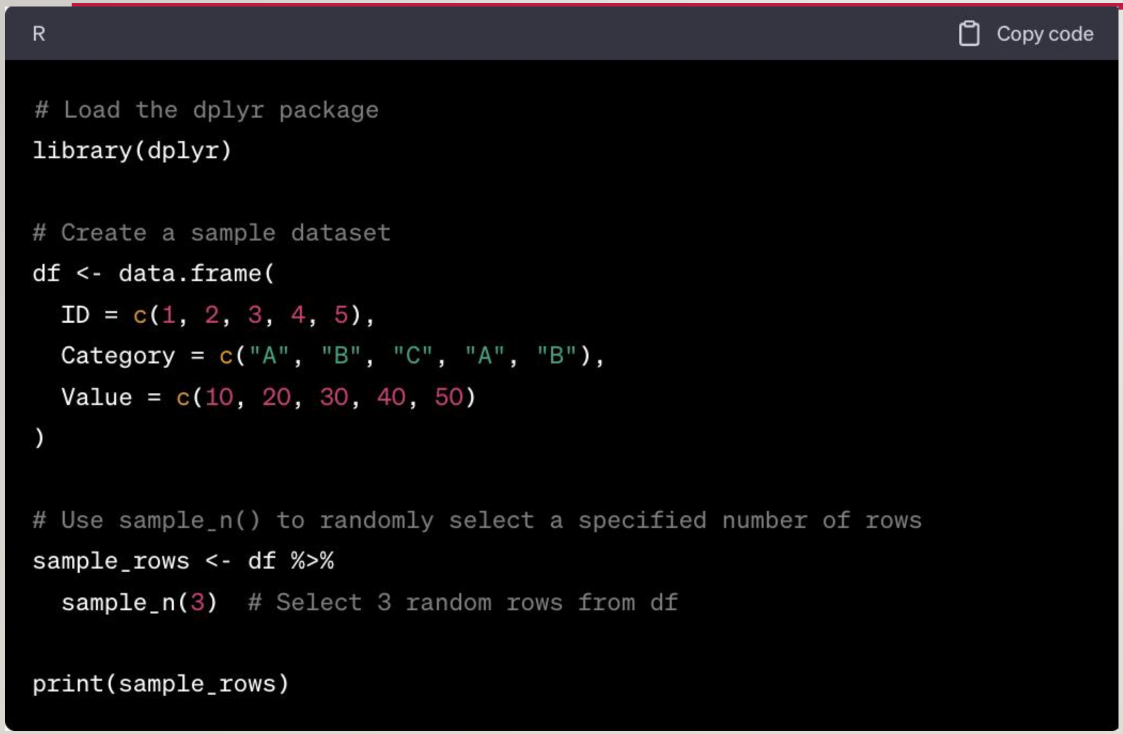
Important dplyr Functions



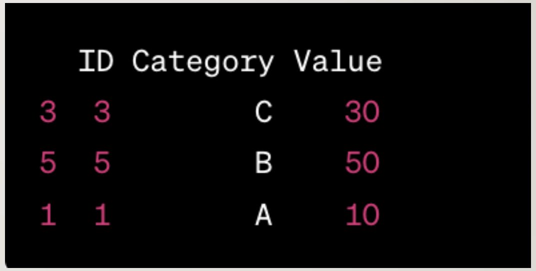
dplyr Function	Description	Equivalent SQL
select()	Selecting columns (variables)	SELECT
filter()	Filter (subset) rows.	WHERE
group_by()	Group the data	GROUP BY
summarise()	Summarise (or aggregate) data	–
arrange()	Sort the data	ORDER BY
join()	Joining data frames (tables)	JOIN
mutate()	Creating New Variables	COLUMN ALIAS

DPLYR PRACTICAL EXAMPLES
EXAMPLE 1 : SELECTING RANDOM N ROWS

- The **sample_n** function selects random rows from a data frame (or table). The second parameter of the function tells R the number of rows to select.
- Example: sample_n(mydata,3)



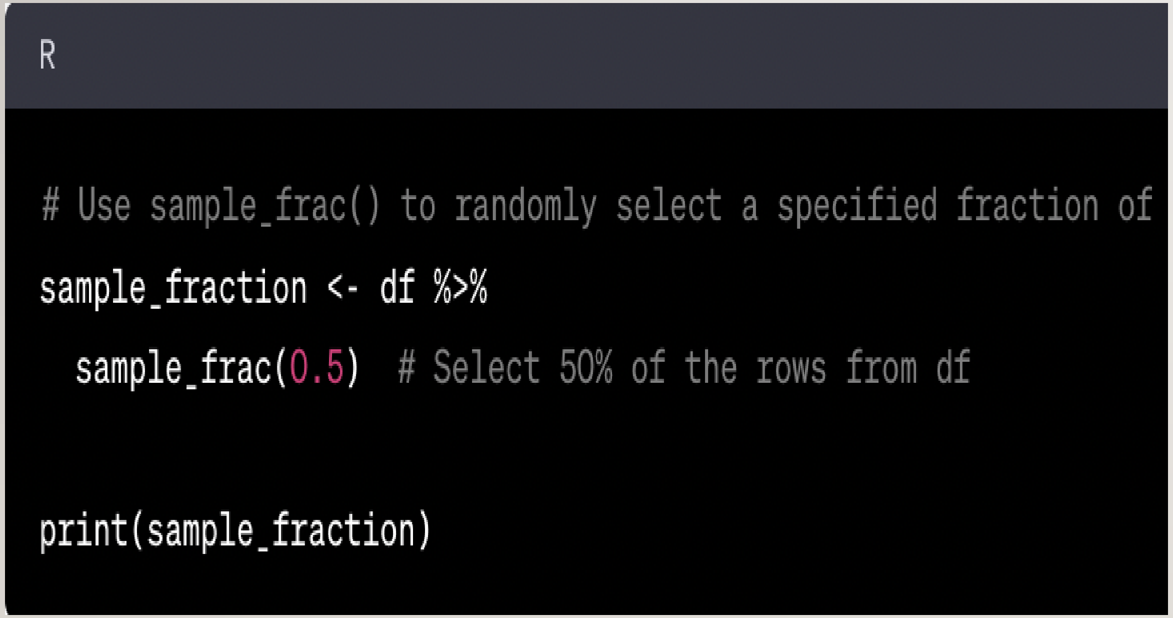
```
# Load the dplyr package
library(dplyr)

# Create a sample dataset
df <- data.frame(
  ID = c(1, 2, 3, 4, 5),
  Category = c("A", "B", "C", "A", "B"),
  Value = c(10, 20, 30, 40, 50)
)

# Use sample_n() to randomly select a specified number of rows
sample_rows <- df %>%
  sample_n(3)  # Select 3 random rows from df

print(sample_rows)
```



```
  ID Category Value
3  3        C    30
5  5        B    50
1  1        A    10
```

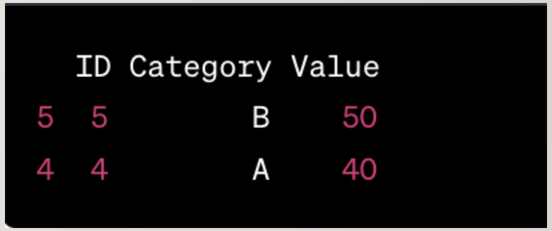
EXAMPLE 2 : SELECTING RANDOM FRACTION OF ROWS

- The **sample_frac** function returns randomly N% of rows. In the example below, it returns randomly 10% of rows.
- Example: sample_frac(mydata,0.1)



R

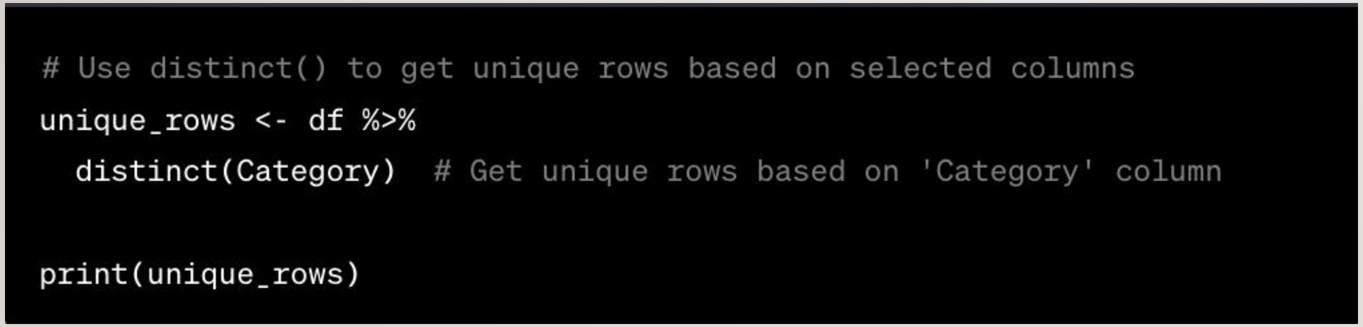
```r
# Use sample_frac() to randomly select a specified fraction of
sample_fraction <- df %>%
  sample_frac(0.5)  # Select 50% of the rows from df

print(sample_fraction)
```



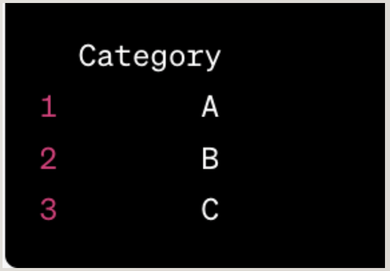
```
  ID Category Value
5  5        B    50
4  4        A    40
```

EXAMPLE3 :REMOVE DUPLICATE ROWS BASED ON ALL THE VARIABLES

- The **distinct function** is used to eliminate duplicates.
- x1 = distinct(mydata)



```
# Use distinct() to get unique rows based on selected columns
unique_rows <- df %>%
  distinct(Category)  # Get unique rows based on 'Category' column

print(unique_rows)
```

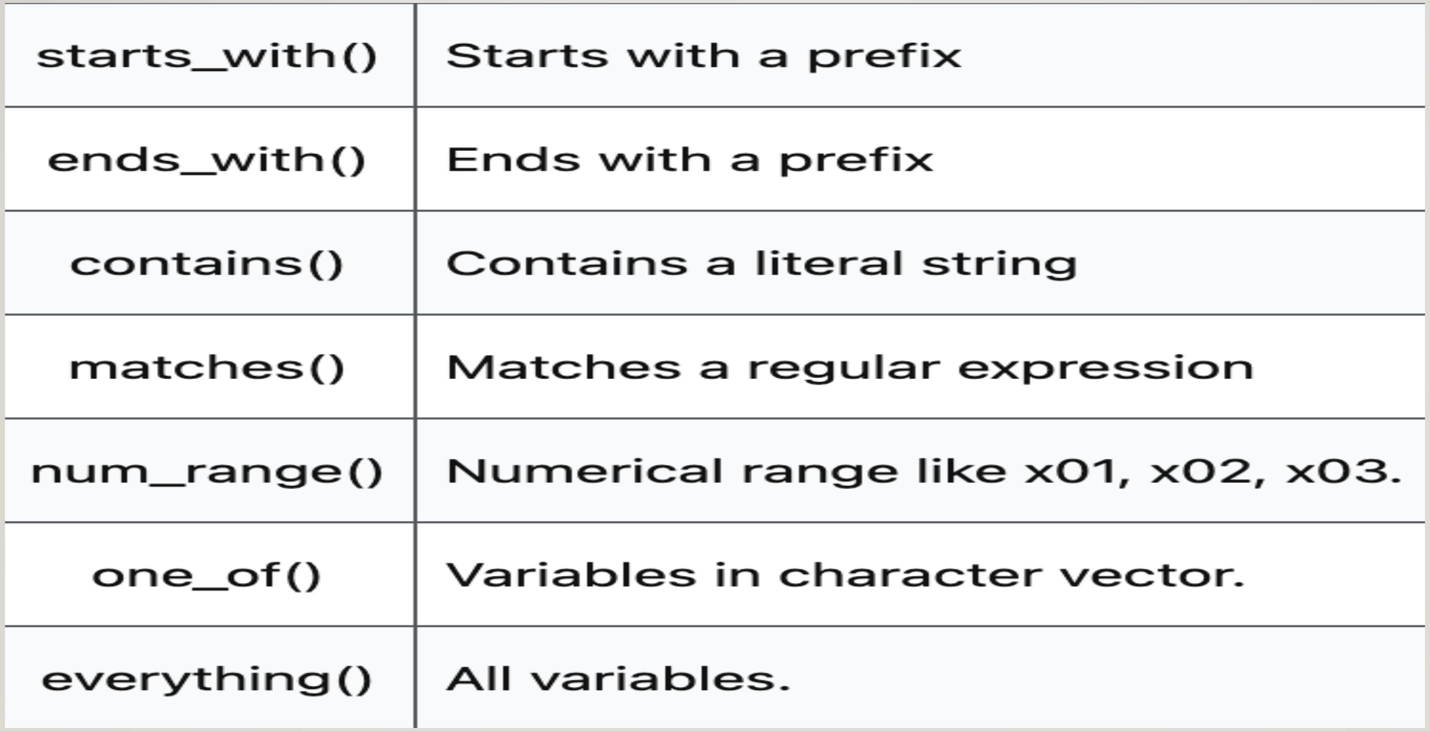


```
  Category
1        A
2        B
3        C
```

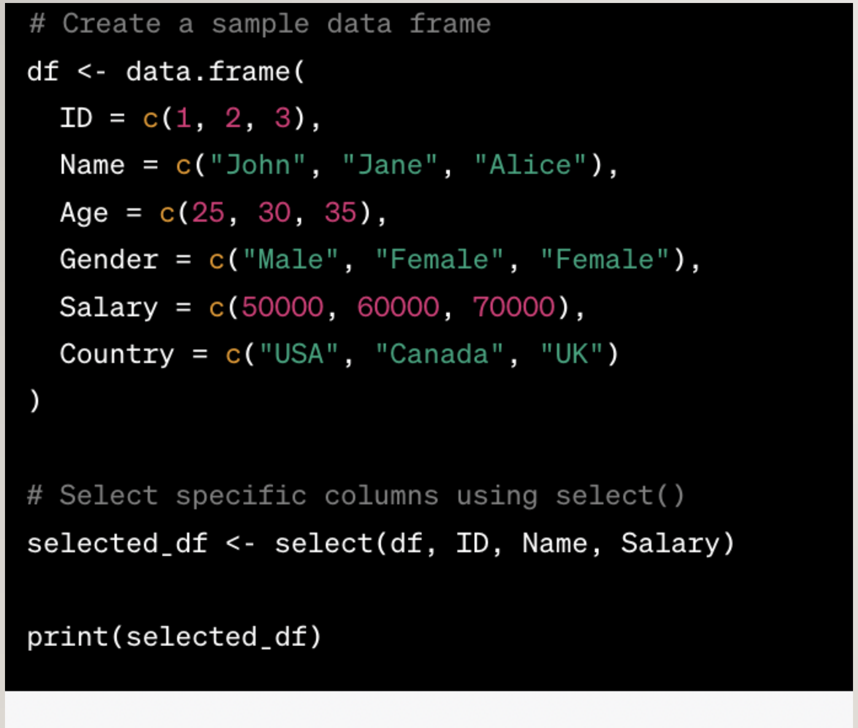
SELECT( ) FUNCTION

- The select() function in R is used to choose specific columns from a data frame. It allows you to create a new data frame with only the columns you specify. Here's an example of how to use the select() function:

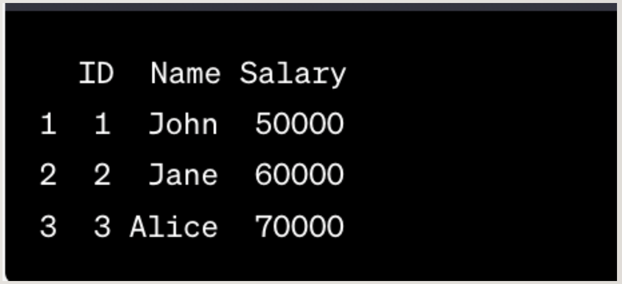
The Following Functions Helps You To Select Variables Based On Their Names.



starts_with()	Starts with a prefix
ends_with()	Ends with a prefix
contains()	Contains a literal string
matches()	Matches a regular expression
num_range()	Numerical range like x01, x02, x03.
one_of()	Variables in character vector.
everything()	All variables.



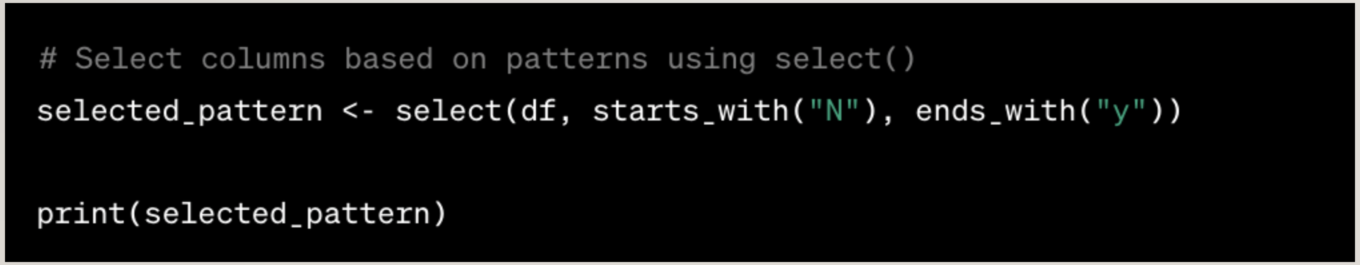
```r
# Create a sample data frame
df <- data.frame(
  ID = c(1, 2, 3),
  Name = c("John", "Jane", "Alice"),
  Age = c(25, 30, 35),
  Gender = c("Male", "Female", "Female"),
  Salary = c(50000, 60000, 70000),
  Country = c("USA", "Canada", "UK")
)

# Select specific columns using select()
selected_df <- select(df, ID, Name, Salary)

print(selected_df)
```



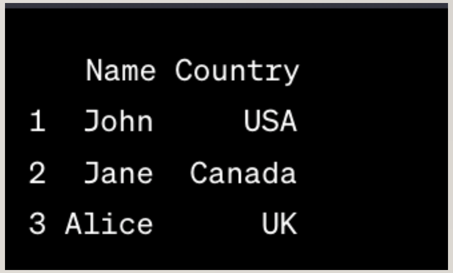
```
  ID  Name Salary
1  1  John  50000
2  2  Jane  60000
3  3 Alice  70000
```

STARTS_WITH(), ENDS_WITH()

- We can also use **select()** with helper functions like **starts_with()**, **ends_with()**, **contains()**, **matches()**, **num_range()**, and **one_of()** to select columns based on specific patterns or conditions.

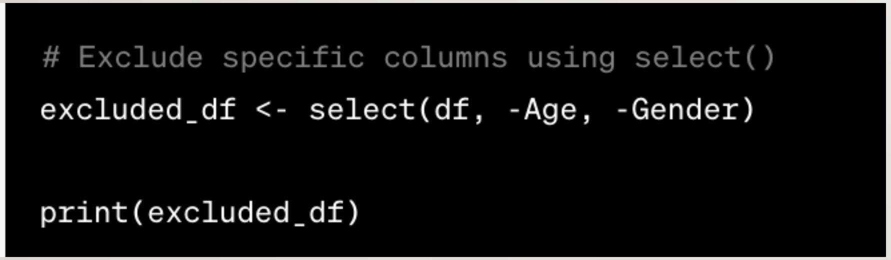


```r
# Select columns based on patterns using select()
selected_pattern <- select(df, starts_with("N"), ends_with("y"))

print(selected_pattern)
```

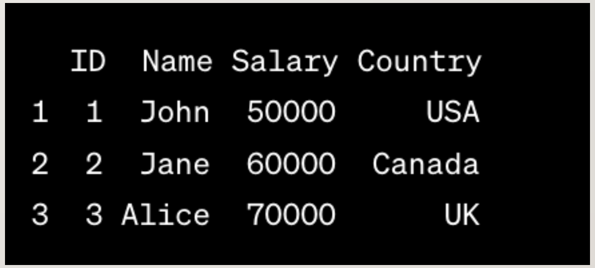


```
   Name Country
1  John     USA
2  Jane  Canada
3 Alice      UK
```

- We can also use the **-** operator to exclude specific columns from the selection.

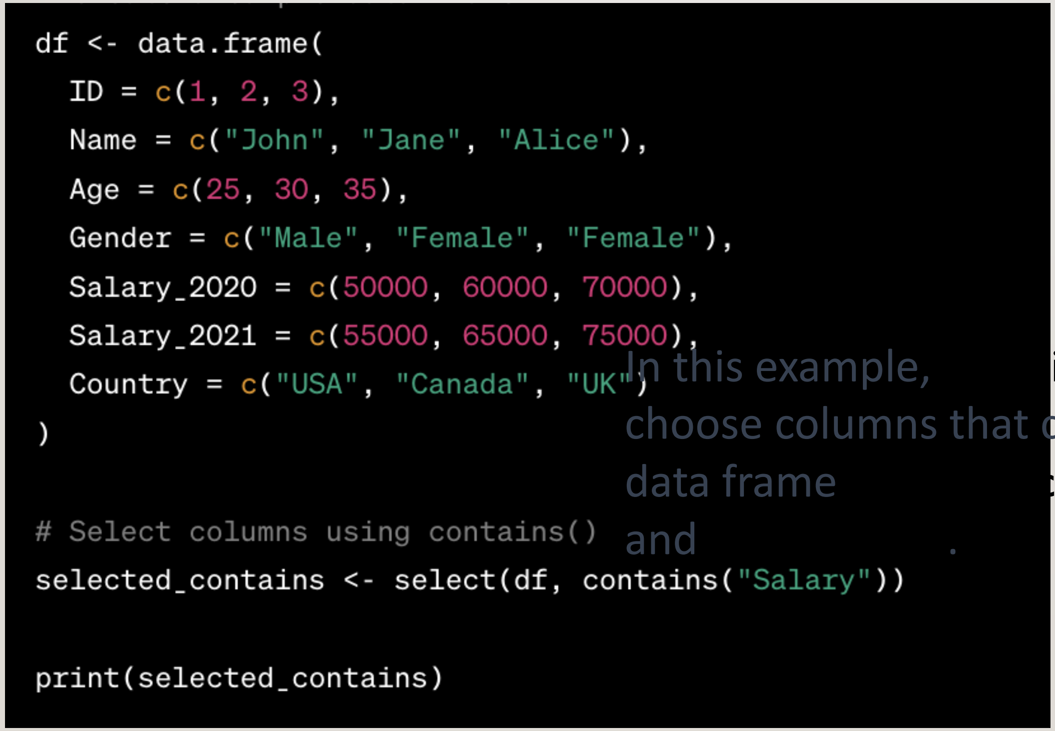


```
# Exclude specific columns using select()
excluded_df <- select(df, -Age, -Gender)

print(excluded_df)
```

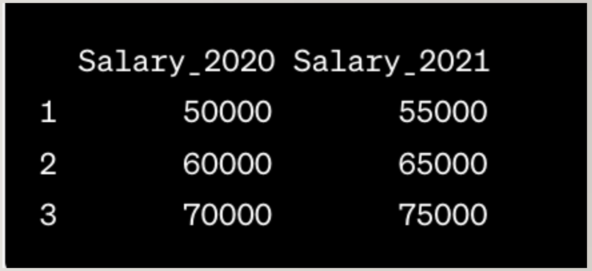


```
  ID  Name Salary Country
1  1  John  50000     USA
2  2  Jane  60000  Canada
3  3 Alice  70000      UK
```

CONTAINS()



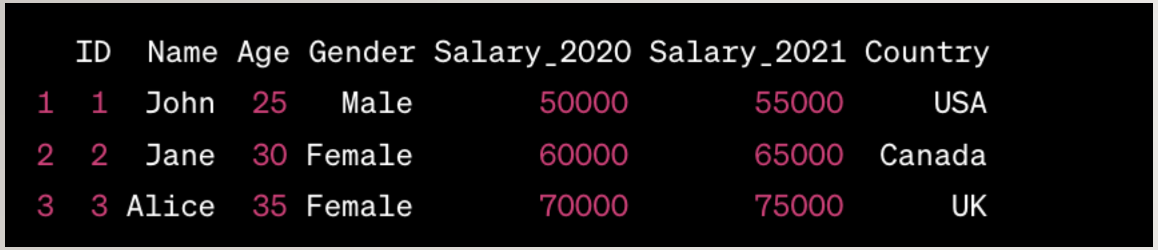
```
df <- data.frame(
  ID = c(1, 2, 3),
  Name = c("John", "Jane", "Alice"),
  Age = c(25, 30, 35),
  Gender = c("Male", "Female", "Female"),
  Salary_2020 = c(50000, 60000, 70000),
  Salary_2021 = c(55000, 65000, 75000),
  Country = c("USA", "Canada", "UK")
)

# Select columns using contains()
selected_contains <- select(df, contains("Salary"))

print(selected_contains)
```



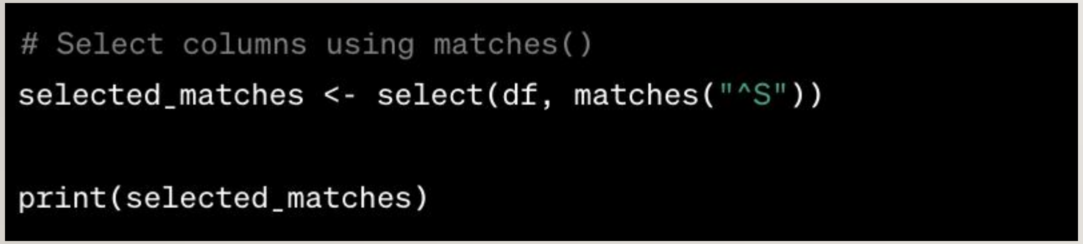
```
  Salary_2020 Salary_2021
1       50000       55000
2       60000       65000
3       70000       75000
```

In this example, ins("Salary") is used with **select()** to choose columns that contain the string "Salary". The resulting data frame ontains includes the columns **Salary_2020** and .

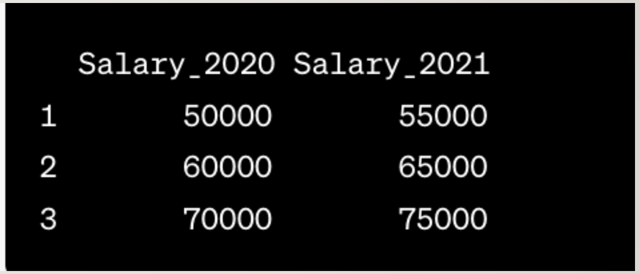
MATCHES()



```
  ID  Name Age Gender Salary_2020 Salary_2021 Country
1  1  John  25   Male       50000       55000     USA
2  2  Jane  30 Female       60000       65000  Canada
3  3 Alice  35 Female       70000       75000      UK
```



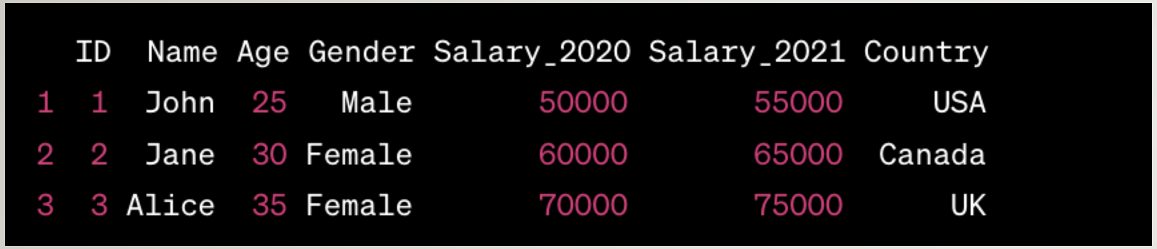
```
# Select columns using matches()
selected_matches <- select(df, matches("^S"))

print(selected_matches)
```



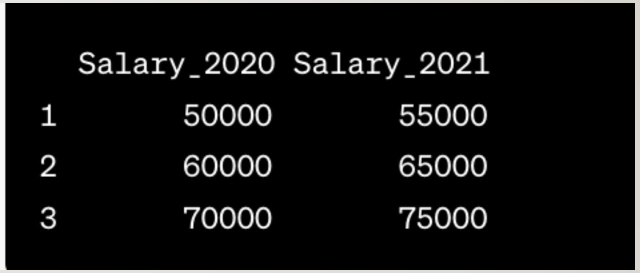
```
  Salary_2020 Salary_2021
1       50000       55000
2       60000       65000
3       70000       75000
```

Here, matches("^S") is used to select columns that start with the letter "S". The resulting data frame selected_matches includes the columns Salary_2020 and Salary_2021.

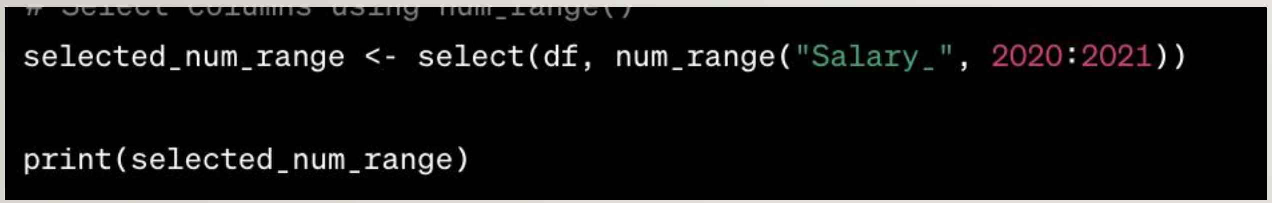
NUM_RANGE()



```
  ID  Name Age Gender Salary_2020 Salary_2021 Country
1  1  John  25   Male       50000       55000     USA
2  2  Jane  30 Female       60000       65000  Canada
3  3 Alice  35 Female       70000       75000      UK
```

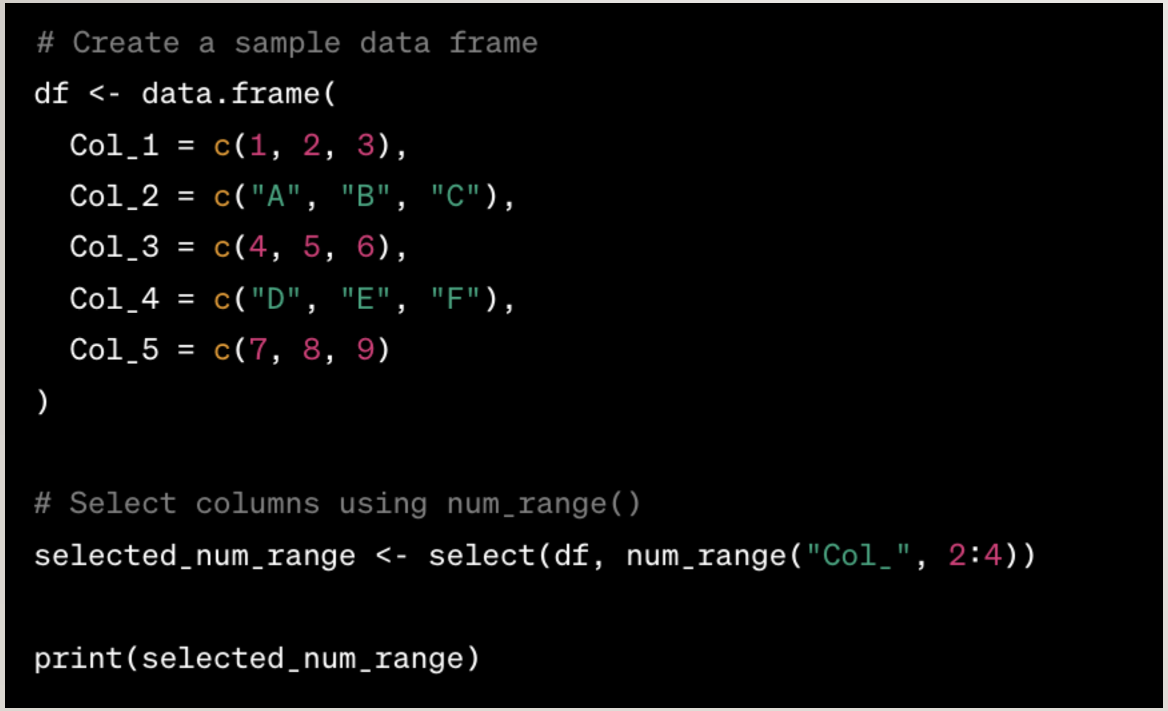


```
  Salary_2020 Salary_2021
1       50000       55000
2       60000       65000
3       70000       75000
```

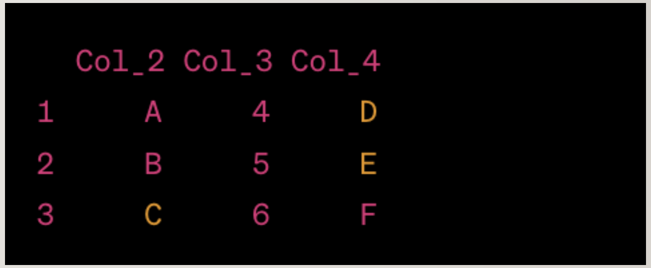


```
# Select columns using num_range()
selected_num_range <- select(df, num_range("Salary_", 2020:2021))

print(selected_num_range)
```

In this example, num_range("Salary_", 2020:2021) is used to select columns with names starting with "Salary_" followed by the years 2020 and 2021. The resulting data frame selected_num_range includes the columns Salary_2020 and Salary_2021.

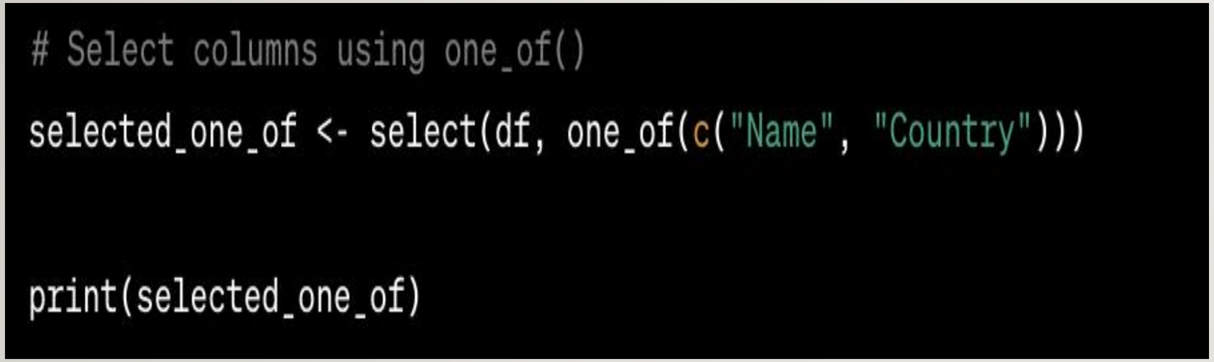


```
# Create a sample data frame
df <- data.frame(
  Col_1 = c(1, 2, 3),
  Col_2 = c("A", "B", "C"),
  Col_3 = c(4, 5, 6),
  Col_4 = c("D", "E", "F"),
  Col_5 = c(7, 8, 9)
)

# Select columns using num_range()
selected_num_range <- select(df, num_range("Col_", 2:4))

print(selected_num_range)
```

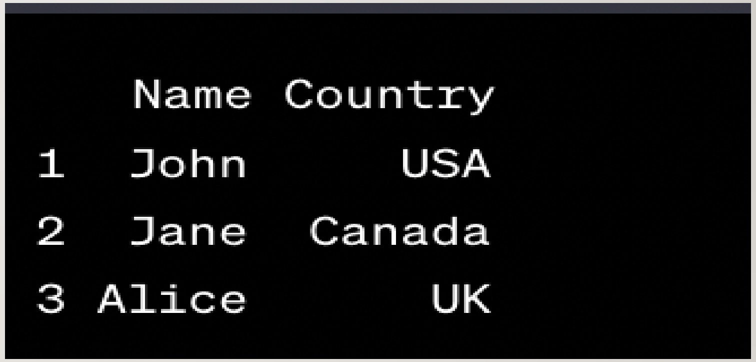


```
  Col_2 Col_3 Col_4
1     A     4     D
2     B     5     E
3     C     6     F
```

ONE_OF



```
# Select columns using one_of()
selected_one_of <- select(df, one_of(c("Name", "Country")))


print(selected_one_of)
```



```
   Name Country
1  John     USA
2  Jane  Canada
3 Alice      UK
```

Here, one_of(c("Name", "Country")) is used to select columns that match either "Name" or "Country". The resulting data frame selected_one_of includes the columns Name and Country

RENAME( ) FUNCTION

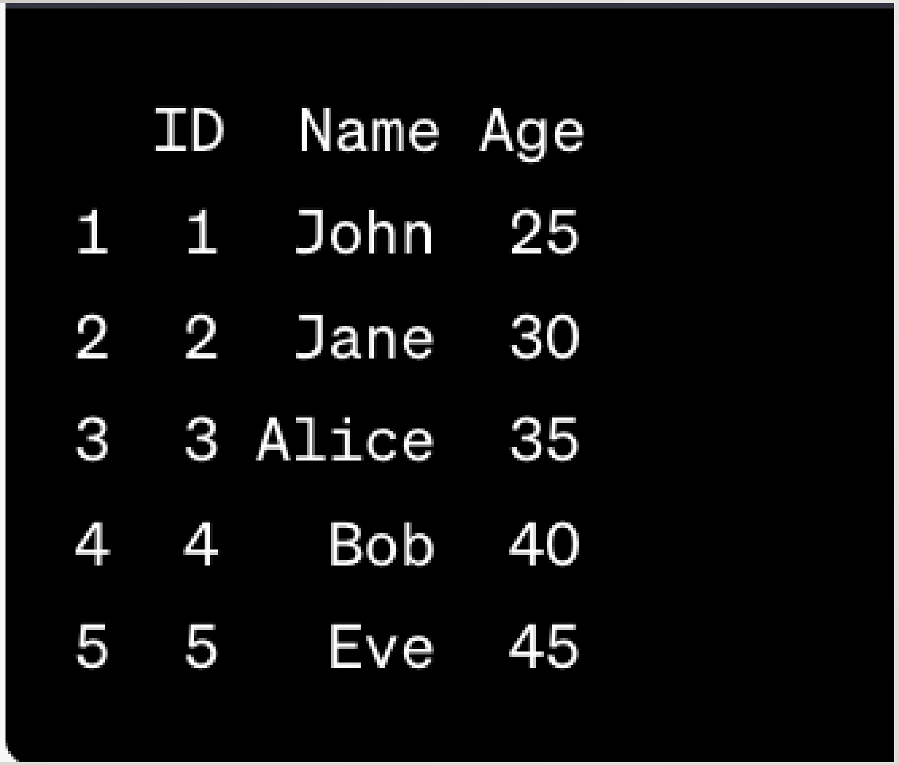
- It is used to change variable name.
- rename() syntax : rename(data , new_name = old_name)
- data : Data Frame
- new_name : New variable name you want to keep
- old_name : Existing Variable Nameo change variable name.

FILTER( ) FUNCTION

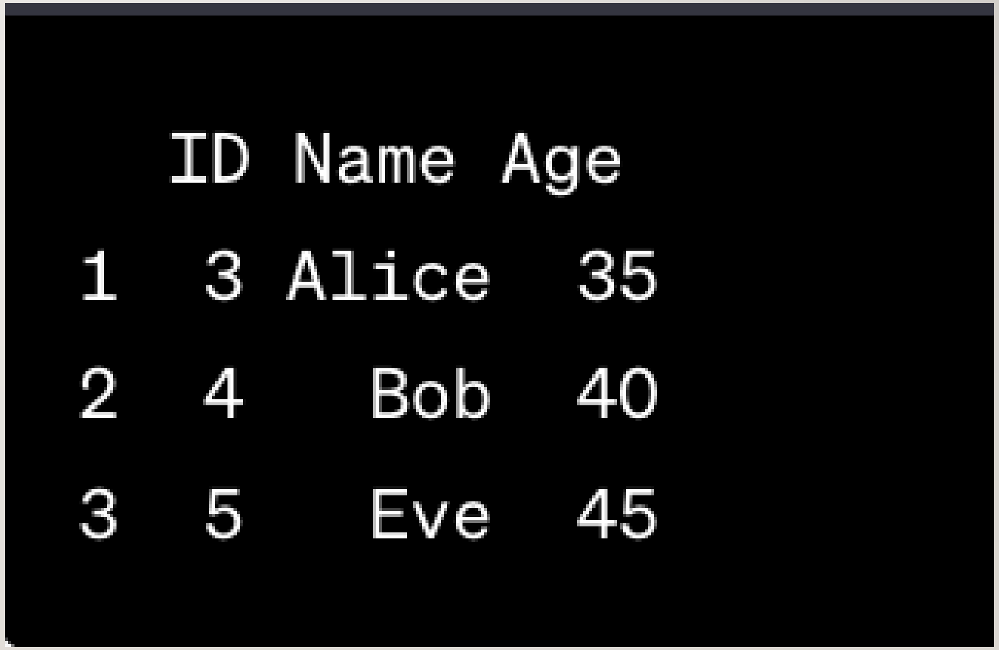
- In R, the **filter()** function is part of the dplyr package and is used to extract rows from a data frame based on specified conditions. It allows you to subset your data based on logical expressions and filter criteria.
- filter() syntax : filter(data , ….)
- data : Data Frame
- …. : Logical Condition

Example 1: Filter Rows Based On A Single Condition

- # Create a data frame
- df <- data.frame(ID = 1:5, Name = c("John", "Jane", "Alice", "Bob", "Eve"), Age = c(25, 30, 35, 40, 45))
- # Filter rows where Age is greater than 30
- filtered_df <- filter(df, Age > 30)
- print(filtered_df)



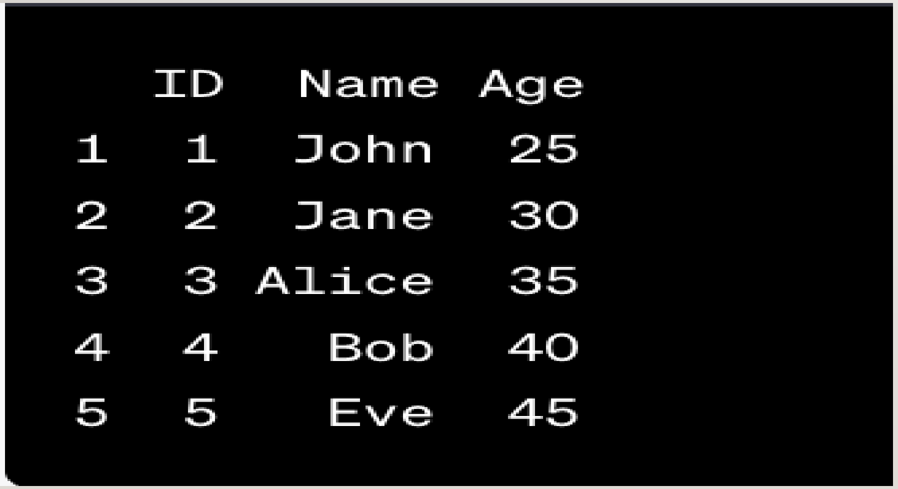
```
  ID  Name Age
1  1  John  25
2  2  Jane  30
3  3 Alice  35
4  4   Bob  40
5  5   Eve  45
```



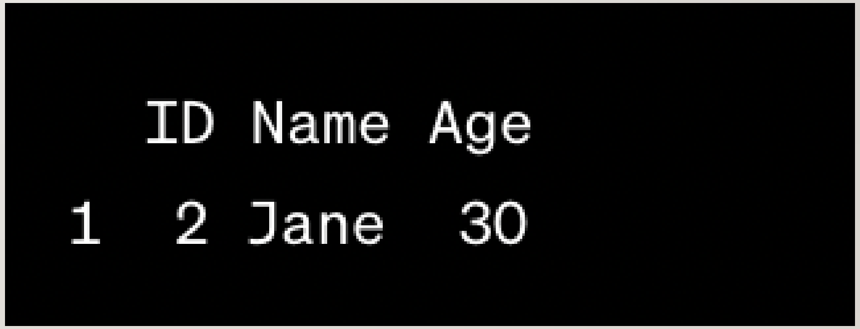
```
  ID  Name Age
1  3 Alice  35
2  4   Bob  40
3  5   Eve  45
```

Example 2: Filter Rows Based On Multiple Conditions

- filtered_df <- filter(df, Age > 30, startsWith(Name, "J"))
  print(filtered_df)



```
   ID  Name Age
1   1  John  25
2   2  Jane  30
3   3 Alice  35
4   4   Bob  40
5   5   Eve  45
```

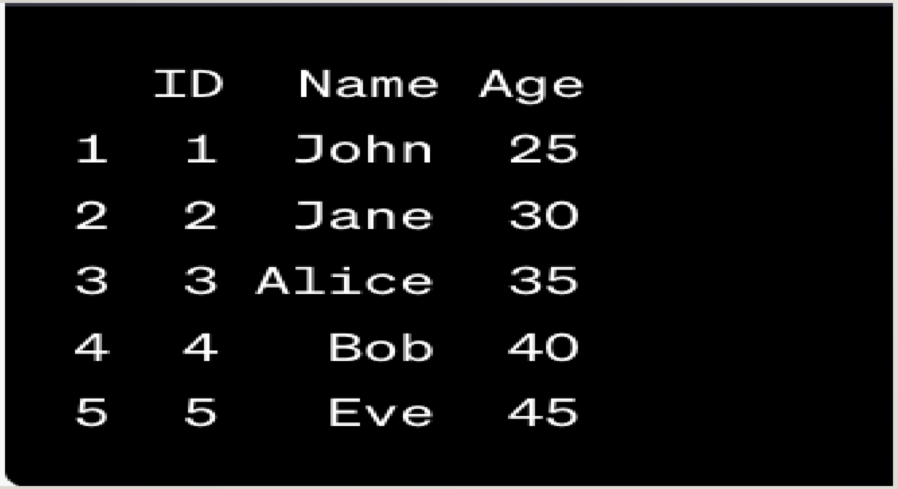


```
  ID Name Age
1  2 Jane  30
```

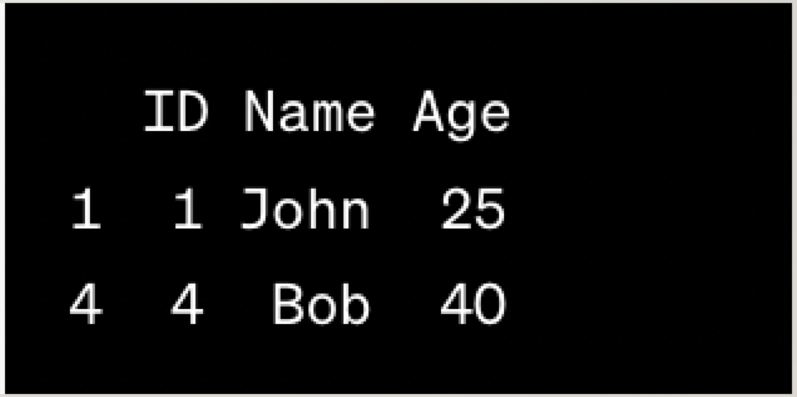
Example 3: Filter Rows Based On Partial String Match

- filtered_df <- filter(df, grepl("o", Name))

print(filtered_df)



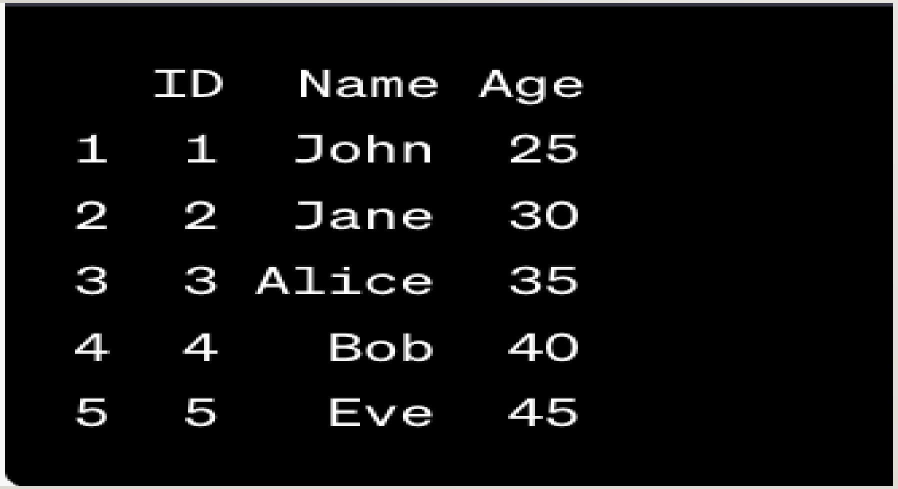
```
   ID  Name Age
1   1  John  25
2   2  Jane  30
3   3 Alice  35
4   4   Bob  40
5   5   Eve  45
```



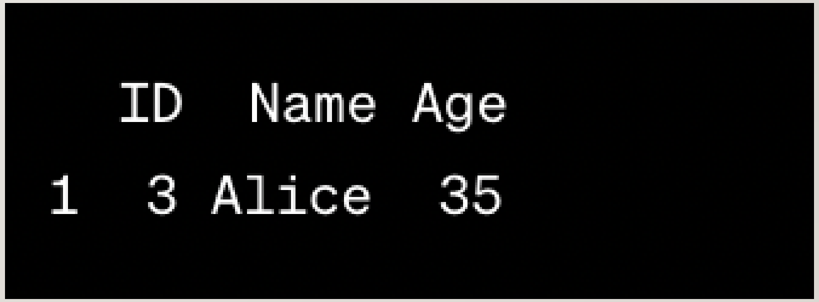
```
  ID Name Age
1  1 John  25
4  4  Bob  40
```

Example 4: Filter Rows Based On A Character Condition

- filtered_df <- filter(df, Name == "Alice")
- print(filtered_df)



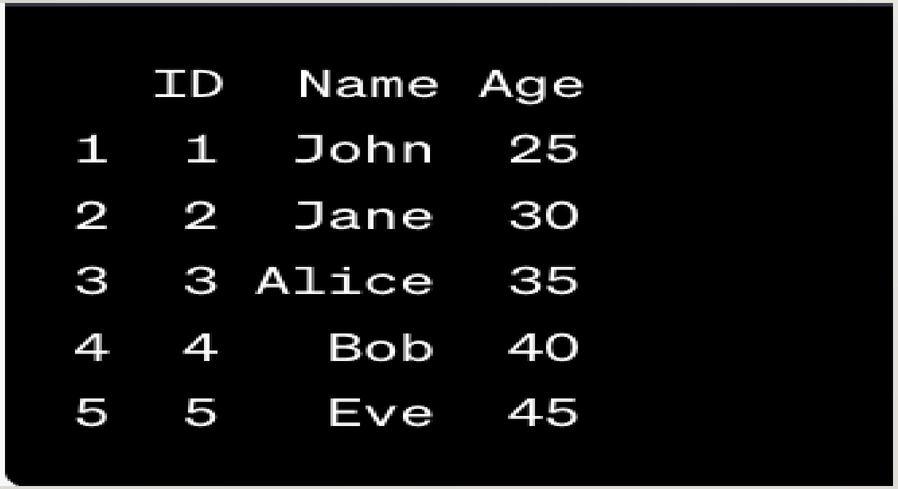
```
  ID  Name Age
1  1  John  25
2  2  Jane  30
3  3 Alice  35
4  4   Bob  40
5  5   Eve  45
```



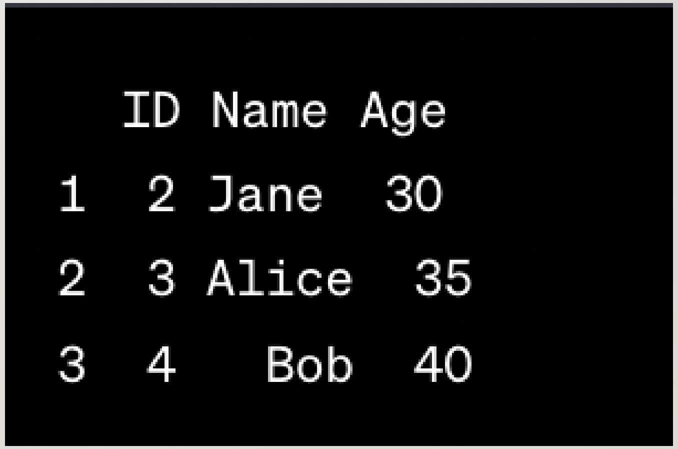
```
  ID  Name Age
1  3 Alice  35
```

Example 5: Filter Rows Based On A Range Of Values

- filtered_df <- filter(df, Age >= 30 & Age <= 40)
  print(filtered_df)



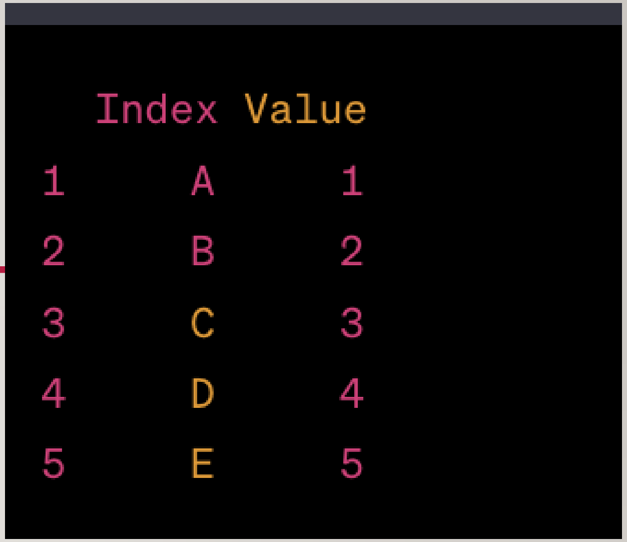
```
  ID  Name Age
1  1  John  25
2  2  Jane  30
3  3 Alice  35
4  4   Bob  40
5  5   Eve  45
```



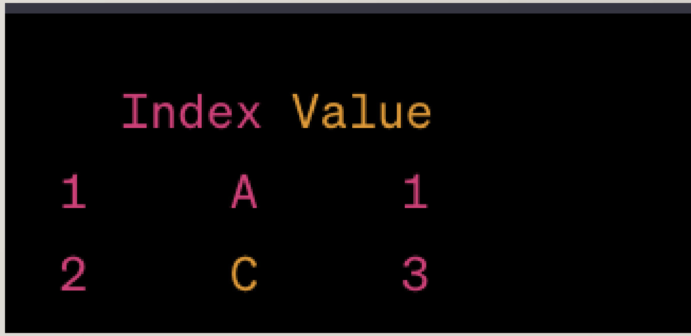
```
  ID  Name Age
1  2  Jane  30
2  3 Alice  35
3  4   Bob  40
```

FILTER USING IN OPERATOR

- library(dplyr)
- # Create a sample data frame
- df <- data.frame(Index = c('A', 'B', 'C', 'D', 'E'), Value = 1:5)
- # Filter rows where 'Index' is either 'A' or 'C’
- filtered_df <- df %>% filter(Index %in% c('A', 'C’))
- # Print the filtered data frame
- filtered_df



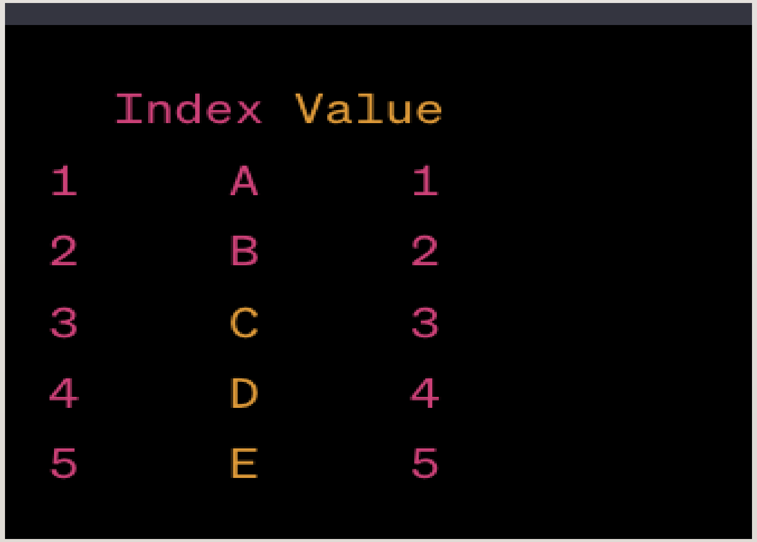
```
  Index Value
1     A     1
2     B     2
3     C     3
4     D     4
5     E     5
```



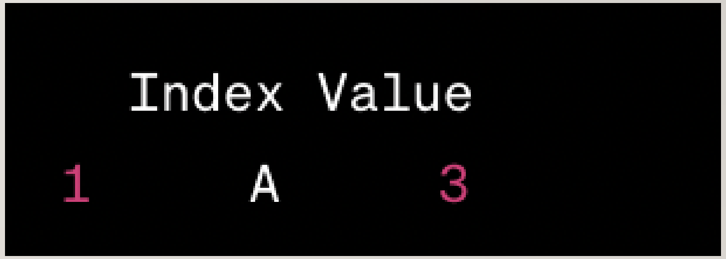
```
  Index Value
1     A     1
2     C     3
```

FILTER USING AND (&)OPERATOR

- # Select rows where 'Index' is 'A' and 'Value' is greater than 2
  filtered_df <- df %>% filter(Index == 'A' & Value > 2)
- # Output
- filtered_df



```
  Index Value
1     A     1
2     B     2
3     C     3
4     D     4
5     E     5
```

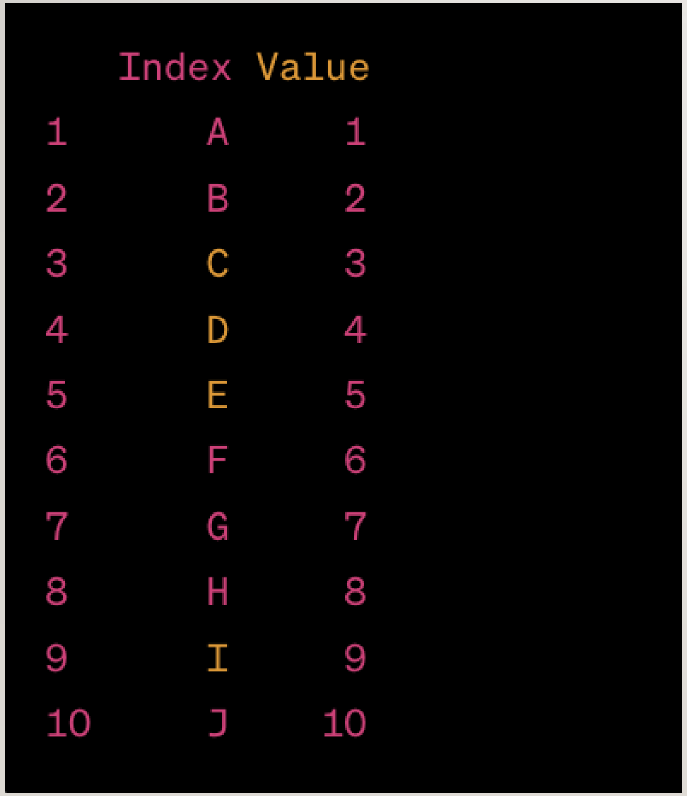


```
  Index Value
1     A     3
```

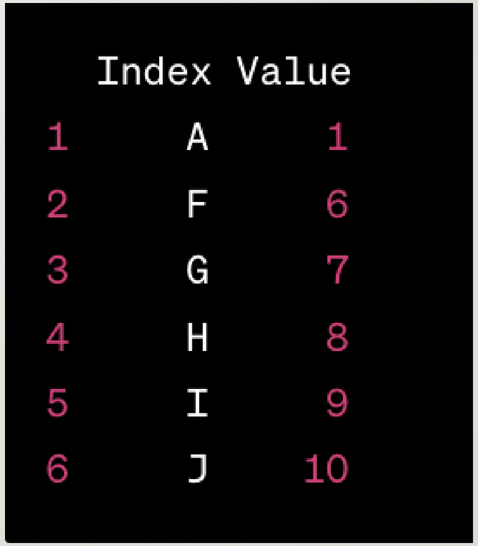
FILTER USING OR (|)OPERATOR

- df <- data.frame(Index = c('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'), Value = 1:10)
- # Select rows where 'Index' is 'A' or 'Value' is greater than 5 filtered_df <- df %>% filter(Index == 'A' | Value > 5) # Output filtered_df

EXAMPLE



```
   Index Value
1      A     1
2      B     2
3      C     3
4      D     4
5      E     5
6      F     6
7      G     7
8      H     8
9      I     9
10     J    10
```



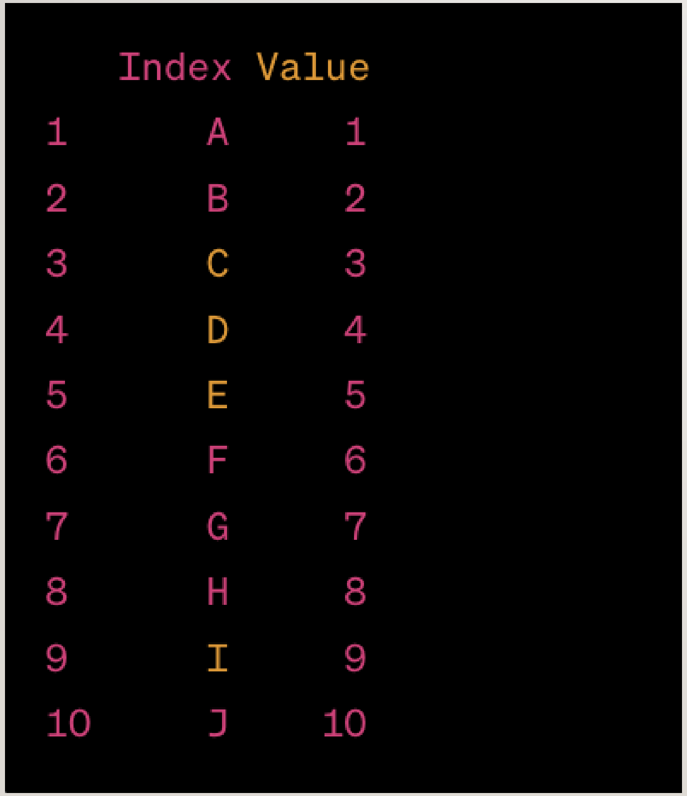
```
   Index Value
1      A     1
2      F     6
3      G     7
4      H     8
5      I     9
6      J    10
```

-
  ```
  library(dplyr)
  ```

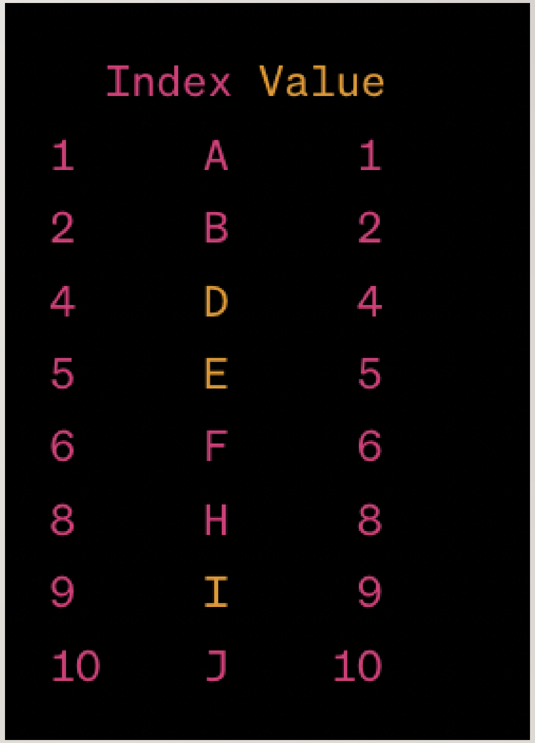
-
  ```
  df <- data.frame( Index = c('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'),
  Value = 1:10 )
  ```

-
  ```
   filtered_df <- df %>% filter(!(Index %in% c('C', 'G')))
  print(filtered_df)
  ```

EXAMPLE



```
      Index  Value
1       A       1
2       B       2
3       C       3
4       D       4
5       E       5
6       F       6
7       G       7
8       H       8
9       I       9
10      J      10
```



```
      Index  Value
1       A       1
2       B       2
4       D       4
5       E       5
6       F       6
8       H       8
9       I       9
10      J      10
```

SUMMARISE() FUNCTION

- The summarise() function in R is a fundamental function from the dplyr package used for data manipulation and summarization. It is commonly used in combination with other functions like group_by() and mutate() to perform data aggregation operations.
- The summarise() function allows you to compute summary statistics or create new variables based on grouped data. It takes a dataset or grouped data frame as input and applies a summarizing operation to each group, resulting in a new data frame with the summarized values.

SYNTAX

- summarise(.data, new_variable = summarizing_operation)
- .data: The input dataset or grouped data frame.
- new_variable: The name of the new variable or column to be created.
- summarizing_operation: The summarizing operation to be applied to the grouped data.

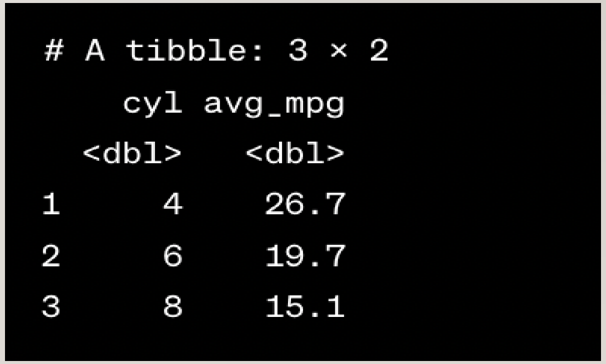
SOME COMMONLY USED SUMMARIZING OPERATIONS INCLUDE:

- mean(): Compute the mean value.
- median(): Compute the median value.
- sum(): Compute the sum of values.
- min(): Find the minimum value.
- max(): Find the maximum value.
- n(): Count the number of observations.

GROUP_BY()

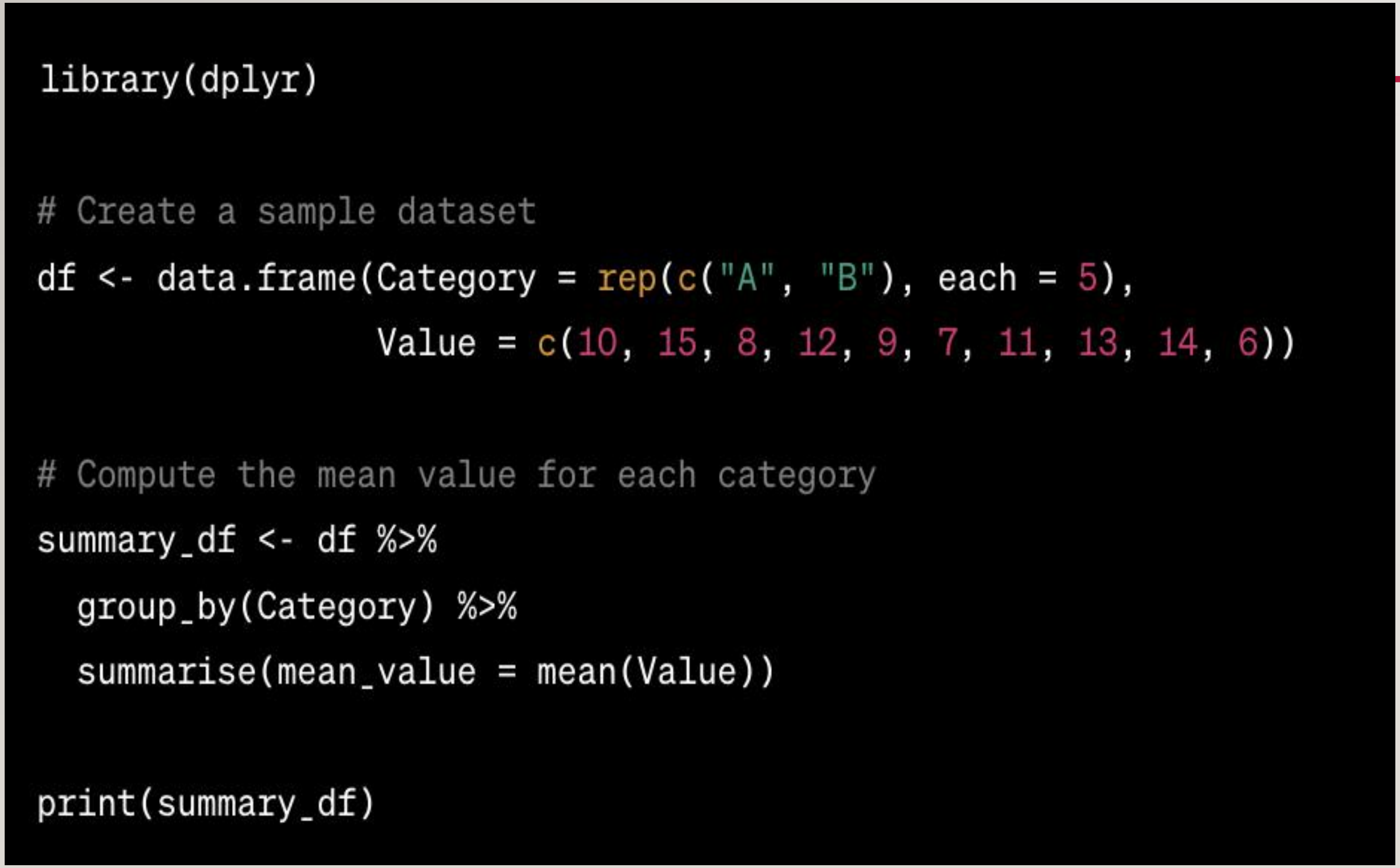
- The group_by() function in R is used to group rows of a dataframe based on one or more variables. Here's an example using the mtcars dataset:

- library(dplyr)
- # Grouping the mtcars dataset by the "cyl" variable
- grouped_data <- group_by(mtcars, cyl)
- # Summarizing the grouped data by calculating the mean of mpg for each cylinder value summary_data <- summarize(grouped_data, avg_mpg = mean(mpg))
- # Printing the summary data
- print(summary_data)

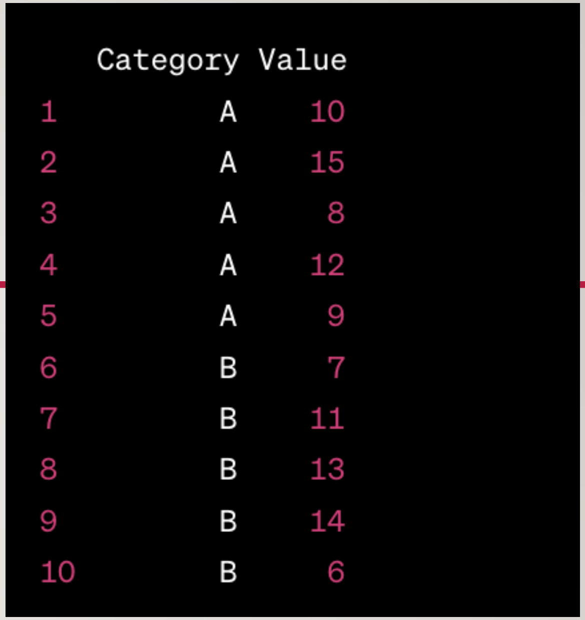


```
# A tibble: 3 × 2
    cyl avg_mpg
  <dbl>   <dbl>
1     4    26.7
2     6    19.7
3     8    15.1
```

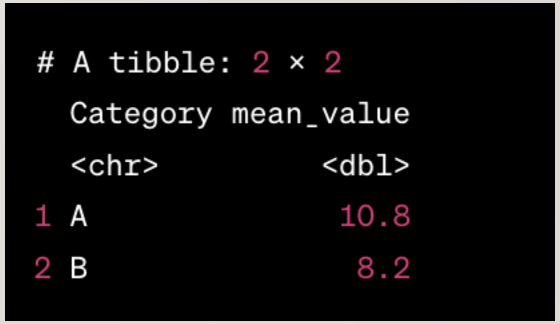
MEAN()



```
library(dplyr)

# Create a sample dataset
df <- data.frame(Category = rep(c("A", "B"), each = 5),
                 Value = c(10, 15, 8, 12, 9, 7, 11, 13, 14, 6))

# Compute the mean value for each category
summary_df <- df %>%
  group_by(Category) %>%
  summarise(mean_value = mean(Value))

print(summary_df)
```

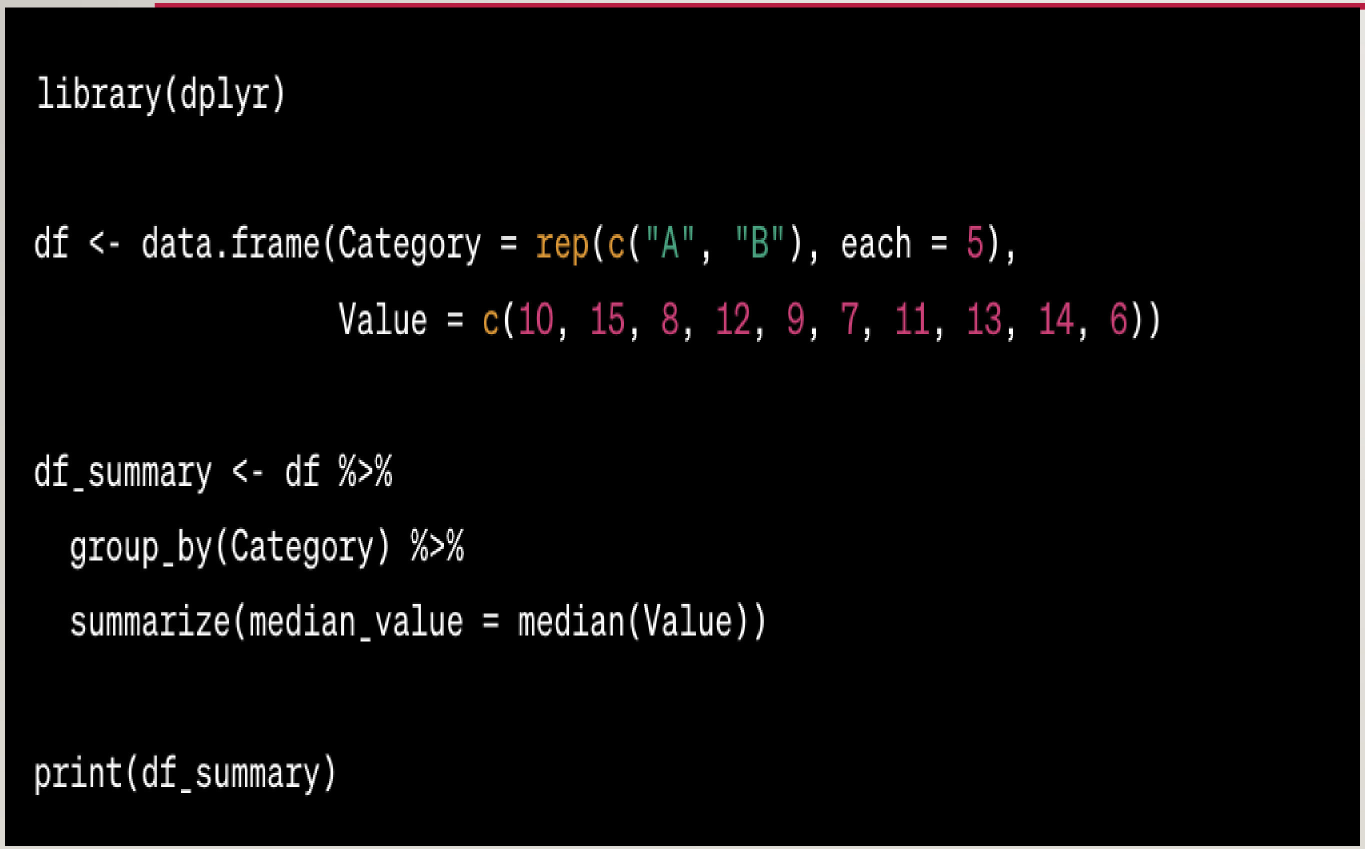


```
   Category Value
1         A    10
2         A    15
3         A     8
4         A    12
5         A     9
6         B     7
7         B    11
8         B    13
9         B    14
10        B     6
```

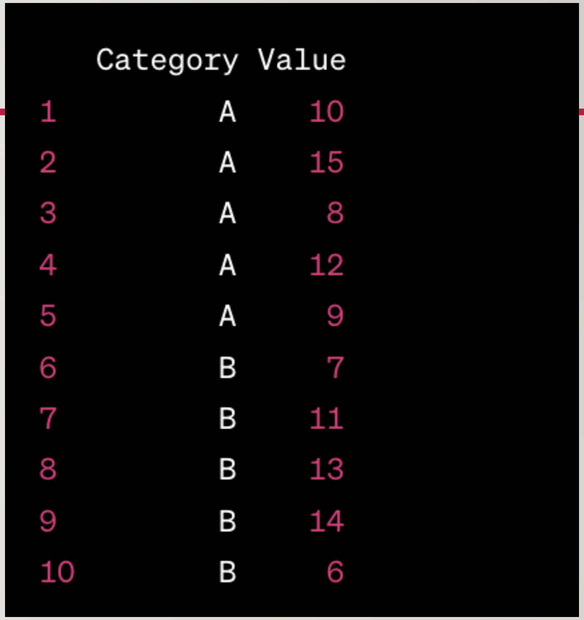


```
# A tibble: 2 × 2
  Category mean_value
  <chr>         <dbl>
1 A              10.8
2 B               8.2
```

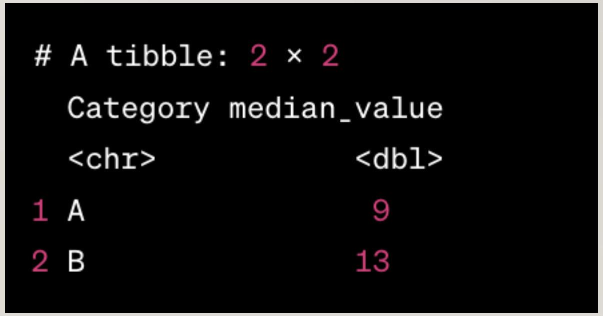
MEDIAN(): COMPUTE THE MEDIAN VALUE.



```
library(dplyr)

df <- data.frame(Category = rep(c("A", "B"), each = 5),
                 Value = c(10, 15, 8, 12, 9, 7, 11, 13, 14, 6))

df_summary <- df %>%
  group_by(Category) %>%
  summarize(median_value = median(Value))

print(df_summary)
```

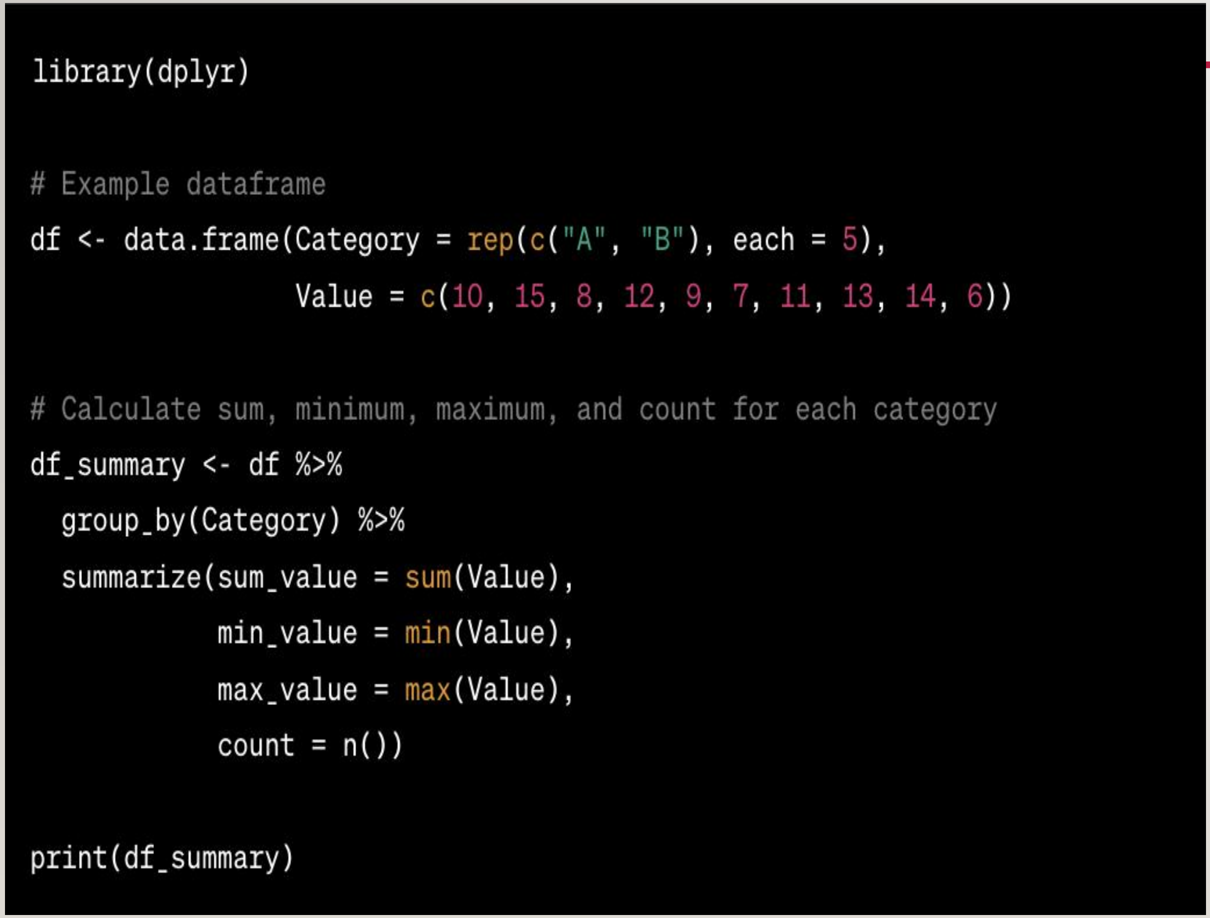


```
   Category Value
1         A    10
2         A    15
3         A     8
4         A    12
5         A     9
6         B     7
7         B    11
8         B    13
9         B    14
10        B     6
```

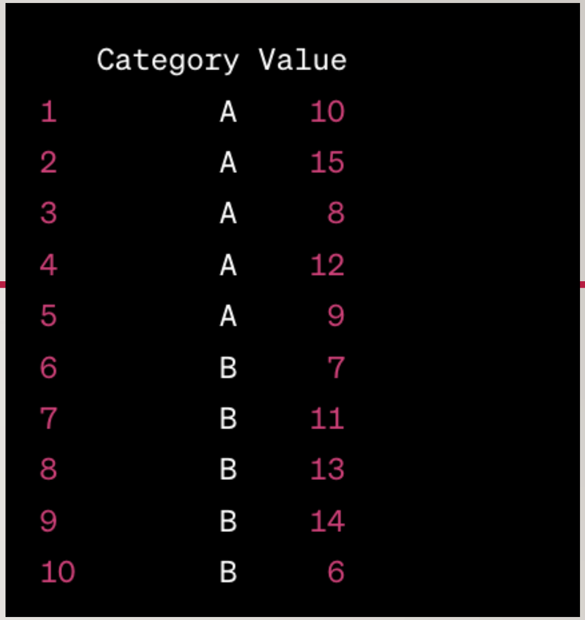


```
# A tibble: 2 × 2
  Category median_value
  <chr>           <dbl>
1 A                   9
2 B                  13
```

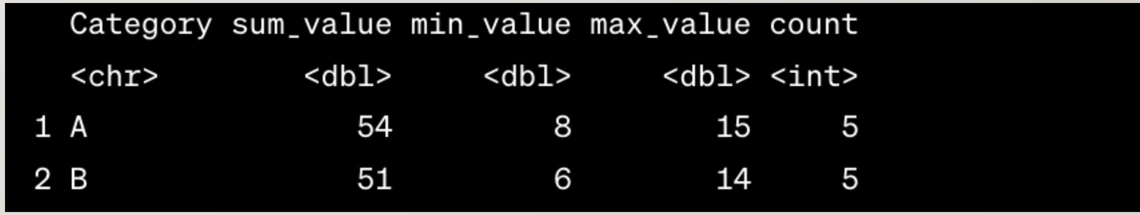
OTHER FUNCTIONS



```
library(dplyr)

# Example dataframe
df <- data.frame(Category = rep(c("A", "B"), each = 5),
                 Value = c(10, 15, 8, 12, 9, 7, 11, 13, 14, 6))

# Calculate sum, minimum, maximum, and count for each category
df_summary <- df %>%
  group_by(Category) %>%
  summarize(sum_value = sum(Value),
            min_value = min(Value),
            max_value = max(Value),
            count = n())

print(df_summary)
```



```
   Category Value
1         A    10
2         A    15
3         A     8
4         A    12
5         A     9
6         B     7
7         B    11
8         B    13
9         B    14
10        B     6
```



```
  Category sum_value min_value max_value count
  <chr>        <dbl>     <dbl>     <dbl> <int>
1 A               54         8        15     5
2 B               51         6        14     5
```

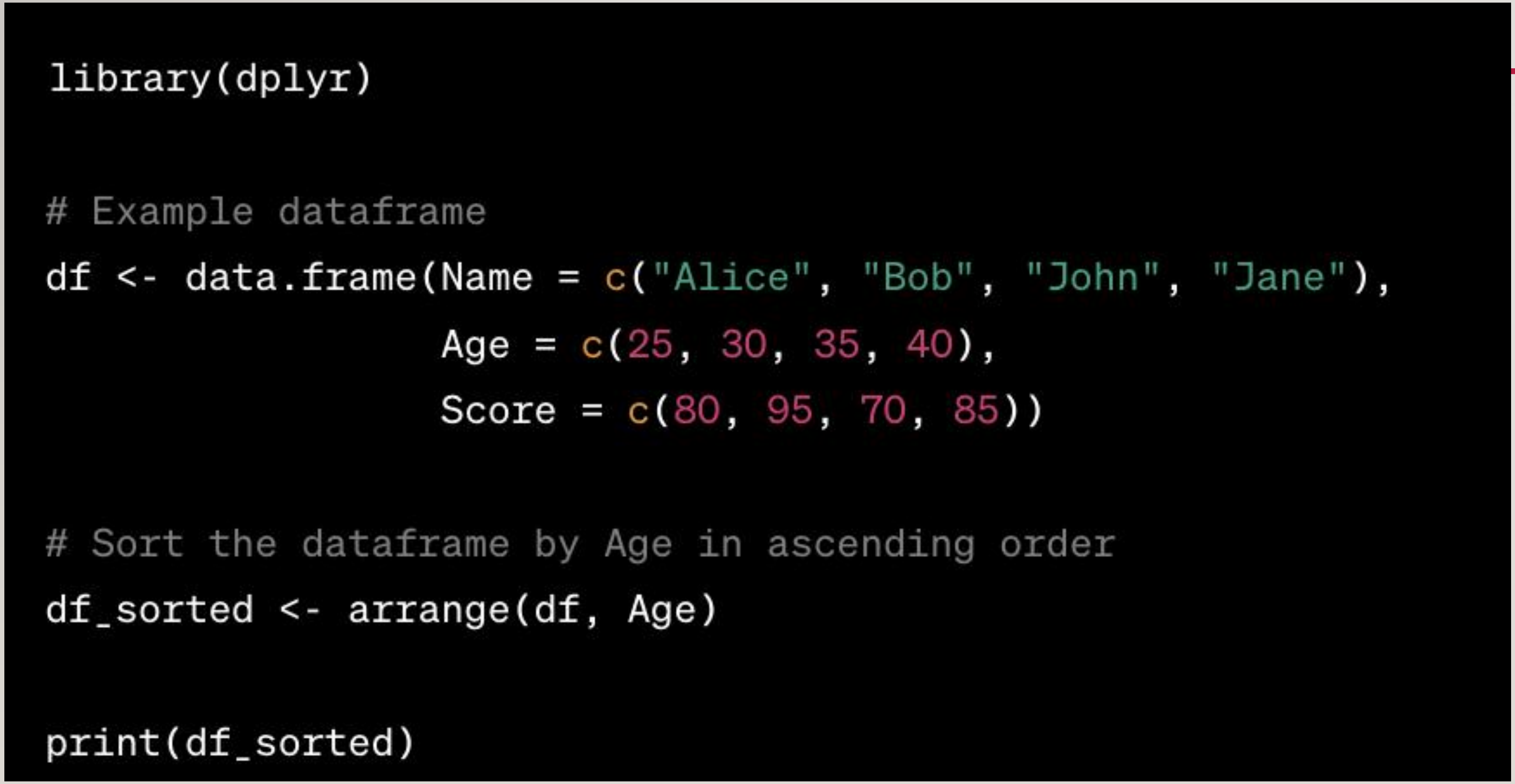
ARRANGE() FUNCTION
It Is Used To Sort The Data

- In R, the arrange() function is used to reorder rows in a dataframe based on one or more columns. It allows you to sort the data in ascending or descending order. The arrange() function is part of the dplyr package and is commonly used for data manipulation and exploratory data analysis.

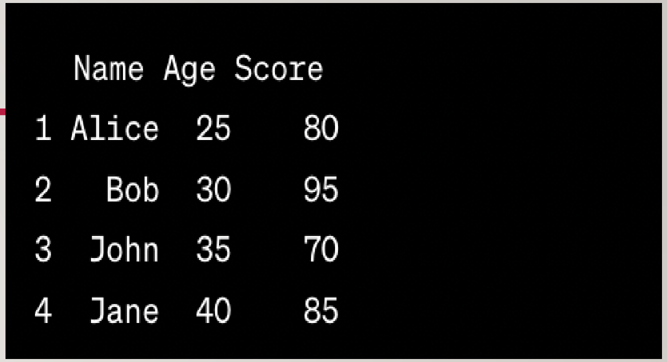
SYNTAX

- arrange(data, column1, column2, ...)
- data: The dataframe or tibble you want to arrange.
- column1, column2, ...: The columns to sort the data by. You can specify multiple columns to sort by. By default, the sorting is done in ascending order.

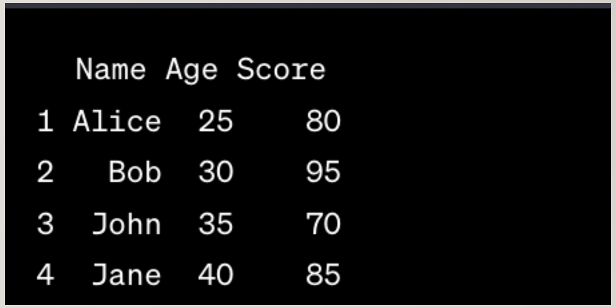
EXAMPLE



```
library(dplyr)

# Example dataframe
df <- data.frame(Name = c("Alice", "Bob", "John", "Jane"),
                 Age = c(25, 30, 35, 40),
                 Score = c(80, 95, 70, 85))

# Sort the dataframe by Age in ascending order
df_sorted <- arrange(df, Age)

print(df_sorted)
```



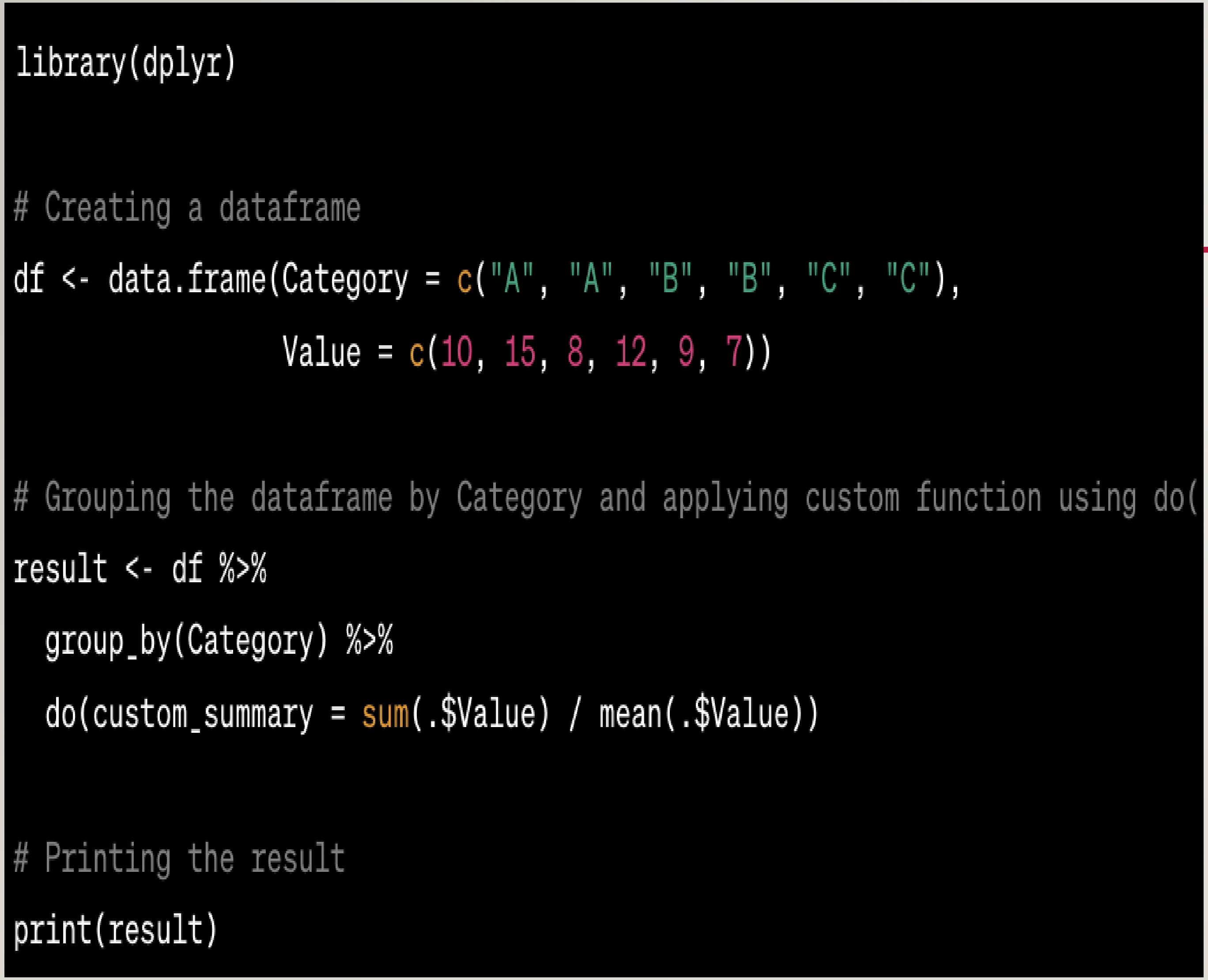
```
   Name Age Score
1 Alice  25    80
2   Bob  30    95
3  John  35    70
4  Jane  40    85
```



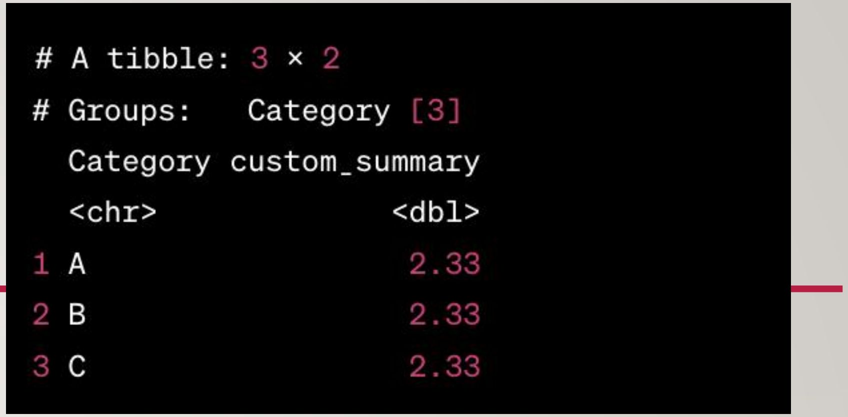
```
   Name Age Score
1 Alice  25    80
2   Bob  30    95
3  John  35    70
4  Jane  40    85
```

DO()

- In R, the do() function is part of the dplyr package and allows you to perform custom operations or transformations on grouped data. It is often used in combination with other dplyr functions like group_by() and summarize(). Here's an example to illustrate its usage:



```
library(dplyr)

# Creating a dataframe
df <- data.frame(Category = c("A", "A", "B", "B", "C", "C"),
                 Value = c(10, 15, 8, 12, 9, 7))

# Grouping the dataframe by Category and applying custom function using do(
result <- df %>%
  group_by(Category) %>%
  do(custom_summary = sum(.$Value) / mean(.$Value))

# Printing the result
print(result)
```

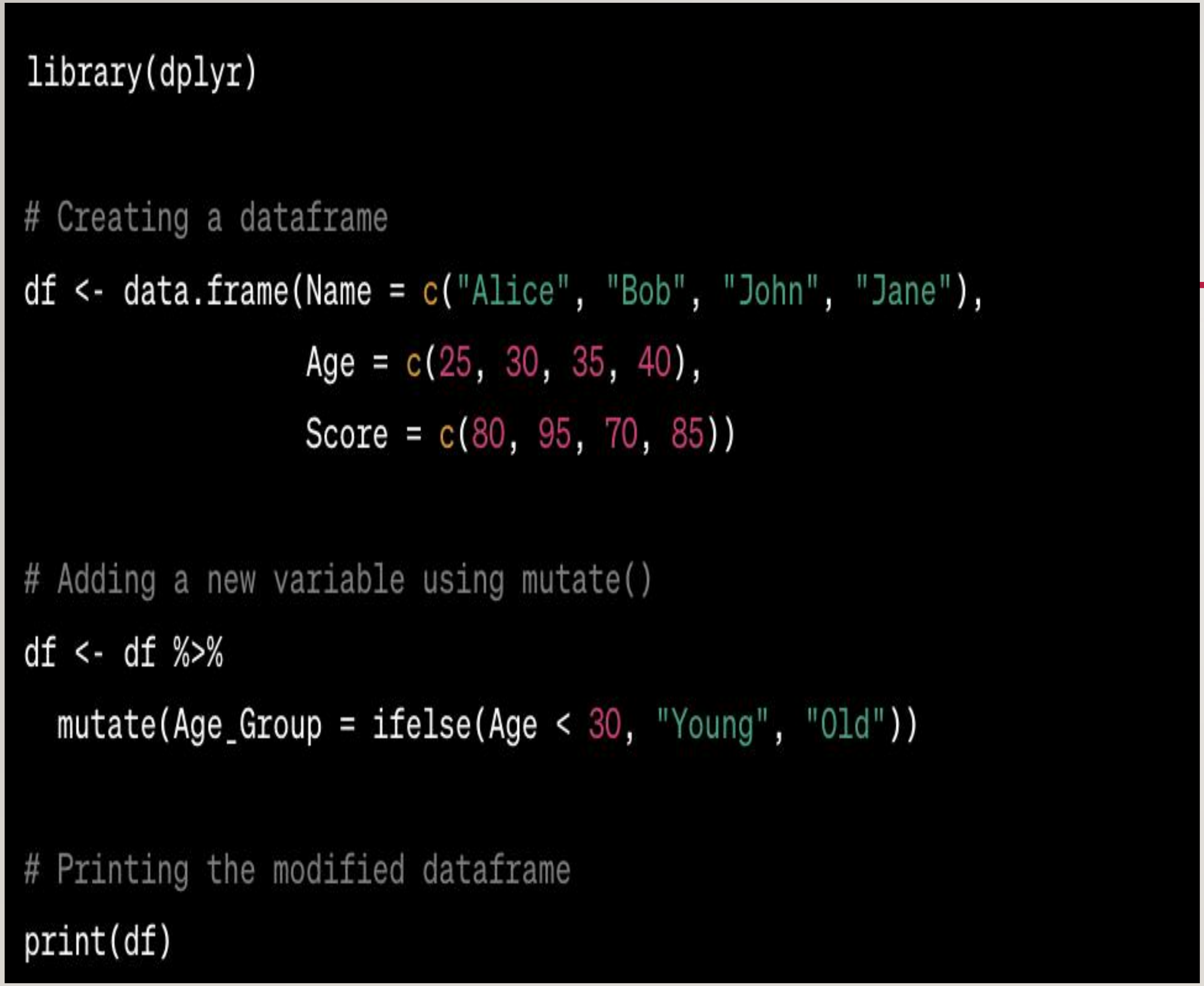


```
# A tibble: 3 × 2
# Groups:   Category [3]
  Category custom_summary
  <chr>             <dbl>
1 A                  2.33
2 B                  2.33
3 C                  2.33
```

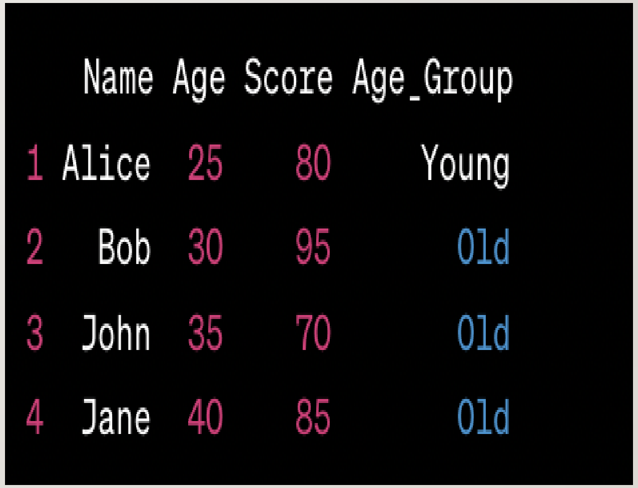
In this example, we grouped the dataframe **df** by the "Category" column using **group_by()**. Then, we used **do()** to apply a custom function that calculates the sum of "Value" divided by the mean of "Value" within each group. The resulting dataframe **result** contains the grouped categories and the corresponding custom summary value for each group.

MUTATE() FUNCTION

- In R, the mutate() function is part of the dplyr package and is used to create or modify variables (columns) within a dataframe. It allows you to perform calculations or transformations on existing variables and create new variables based on those calculations. Here's an example to illustrate its usage:



```r
library(dplyr)


# Creating a dataframe
df <- data.frame(Name = c("Alice", "Bob", "John", "Jane"),
                 Age = c(25, 30, 35, 40),
                 Score = c(80, 95, 70, 85))


# Adding a new variable using mutate()
df <- df %>%
  mutate(Age_Group = ifelse(Age < 30, "Young", "Old"))


# Printing the modified dataframe
print(df)
```

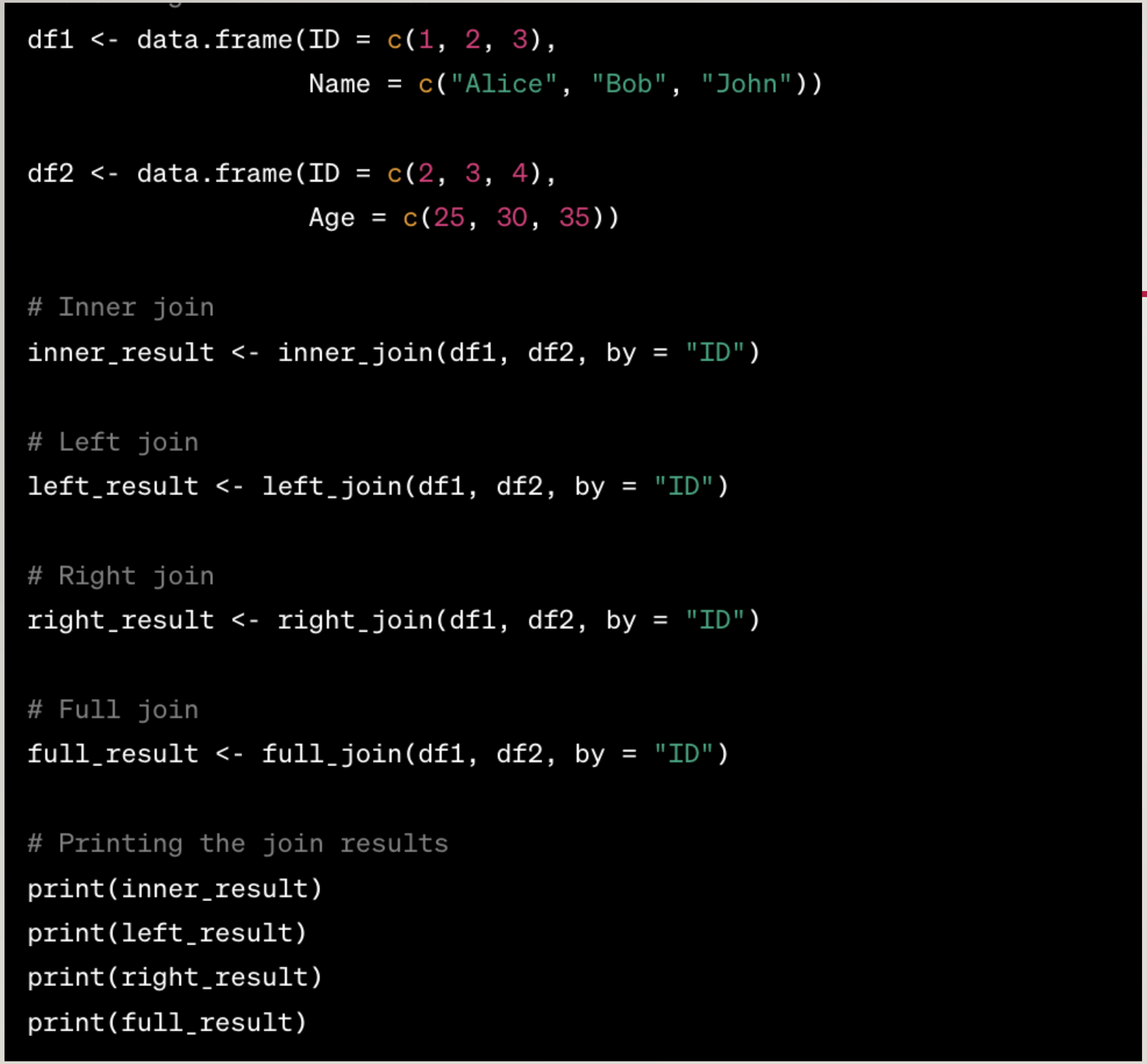


```
   Name Age Score Age_Group
1 Alice  25    80     Young
2   Bob  30    95       Old
3  John  35    70       Old
4  Jane  40    85       Old
```

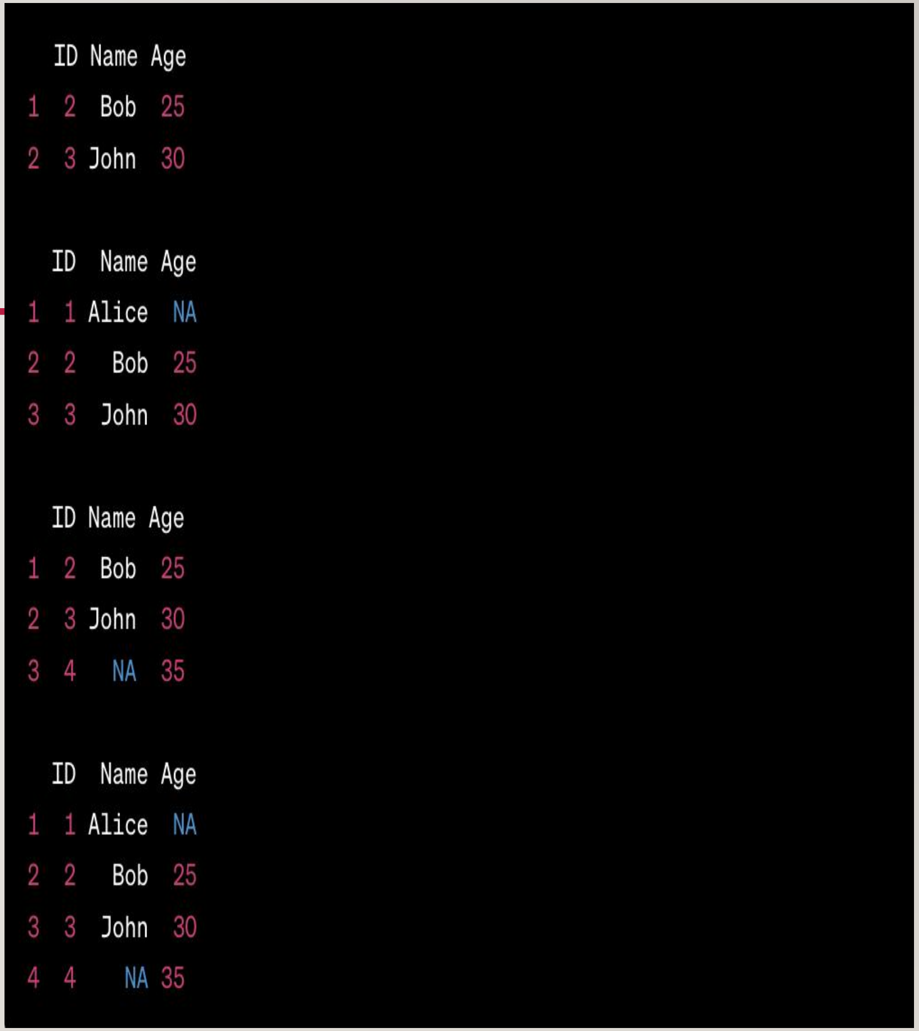
JOIN() FUNCTION

In R, the join() function is used to combine two or more data frames based on a common variable or set of variables. It is part of the dplyr package and provides several types of joins, including inner join, left join, right join, and full join. Here's an overview of the different join functions available in dplyr:

1. Inner Join (inner_join()): Returns only the rows that have matching values in both data frames.
2. Left Join (left_join()): Returns all the rows from the left data frame and the matched rows from the right data frame. If there is no match, it includes NA values for the right data frame columns.
3. Right Join (right_join()): Returns all the rows from the right data frame and the matched rows from the left data frame. If there is no match, it includes NA values for the left data frame columns.
4. Full Join (full_join()): Returns all the rows from both data frames. If there is no match, it includes NA values for the unmatched columns



```r
df1 <- data.frame(ID = c(1, 2, 3),
                  Name = c("Alice", "Bob", "John"))

df2 <- data.frame(ID = c(2, 3, 4),
                  Age = c(25, 30, 35))

# Inner join
inner_result <- inner_join(df1, df2, by = "ID")

# Left join
left_result <- left_join(df1, df2, by = "ID")

# Right join
right_result <- right_join(df1, df2, by = "ID")

# Full join
full_result <- full_join(df1, df2, by = "ID")

# Printing the join results
print(inner_result)
print(left_result)
print(right_result)
print(full_result)
```



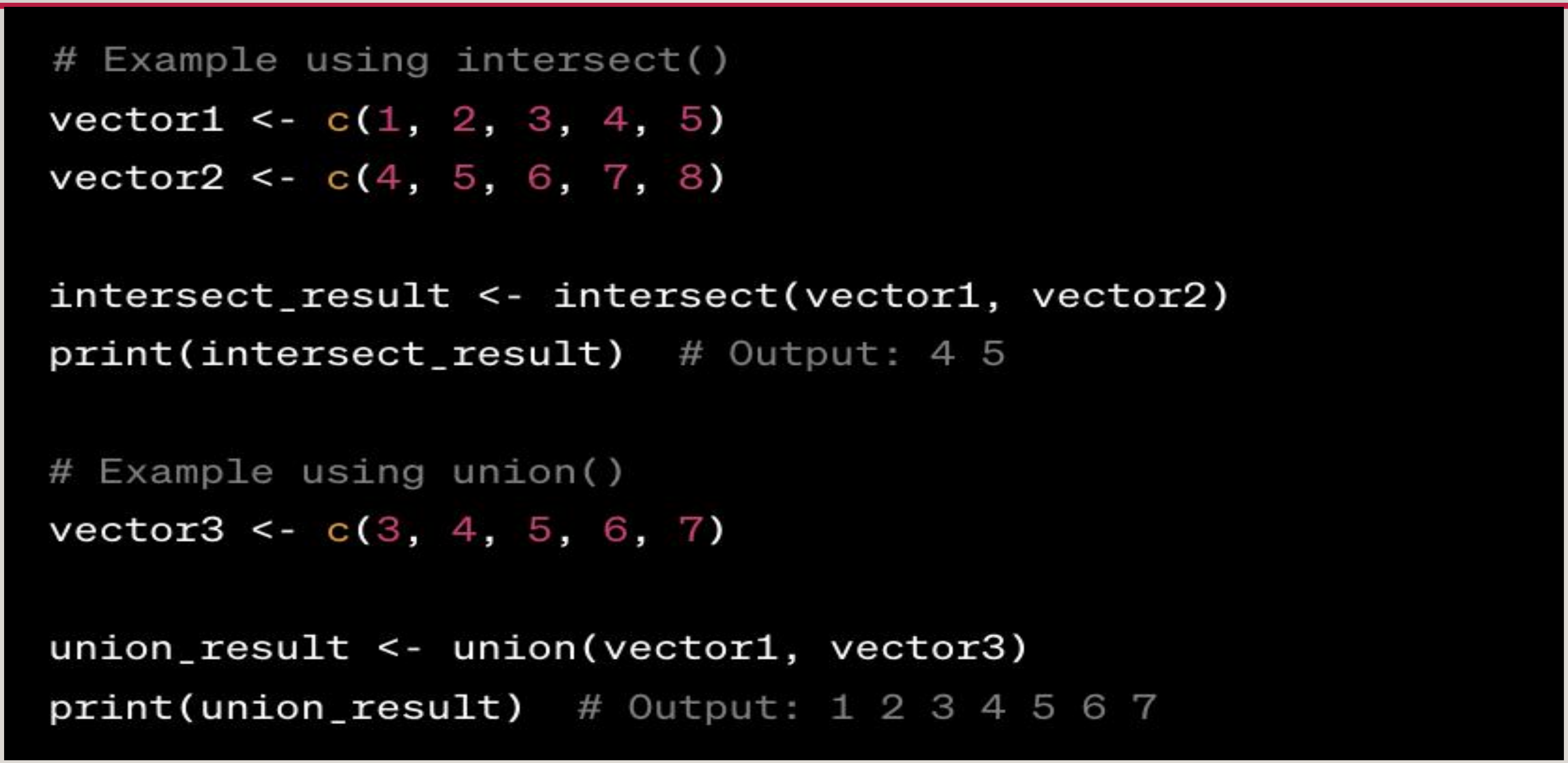
```
  ID Name Age
1  2  Bob  25
2  3 John  30

  ID  Name Age
1  1 Alice  NA
2  2   Bob  25
3  3  John  30

  ID Name Age
1  2  Bob  25
2  3 John  30
3  4   NA  35

  ID  Name Age
1  1 Alice  NA
2  2   Bob  25
3  3  John  30
4  4    NA 35
```

INTERSECT() AND UNION()

- In R, the intersect() and union() functions are used to perform set operations on vectors or data frames. Here's an explanation of each function:

1. intersect(x, y): Returns a vector or data frame containing the common elements between two input vectors or data frames x and y. The resulting elements are unique and appear in the order of x. If the input arguments are data frames, the columns must have the same names and types.
2. union(x, y): Returns a vector or data frame containing all the unique elements from both input vectors or data frames x and y. The resulting elements appear in the order they first appear in the combined vectors. If the input arguments are data frames, the columns must have the same names and types.

EXAMPLE

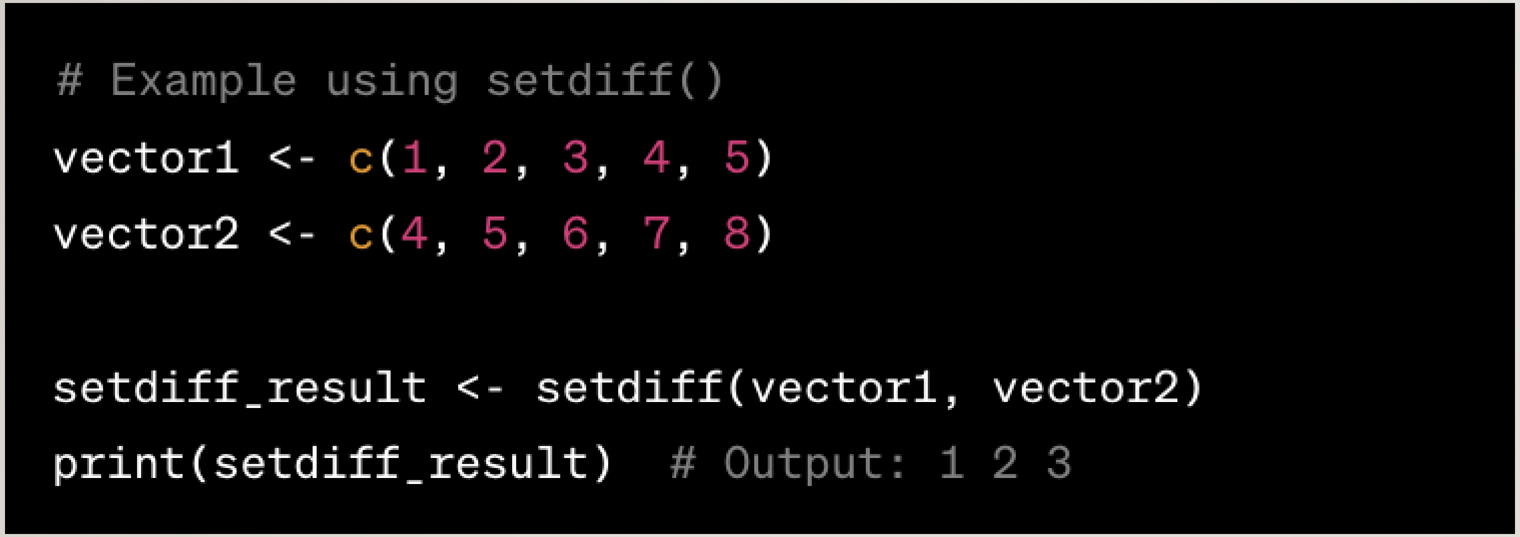


```
# Example using intersect()
vector1 <- c(1, 2, 3, 4, 5)
vector2 <- c(4, 5, 6, 7, 8)

intersect_result <- intersect(vector1, vector2)
print(intersect_result)  # Output: 4 5

# Example using union()
vector3 <- c(3, 4, 5, 6, 7)

union_result <- union(vector1, vector3)
print(union_result)  # Output: 1 2 3 4 5 6 7
```

SETDIFF(

- In R, the setdiff() function is used to find the set difference between two vectors or data frames. It returns the elements from the first input that are not present in the second input.
- Here's the syntax of the setdiff() function:

EXAMPLE



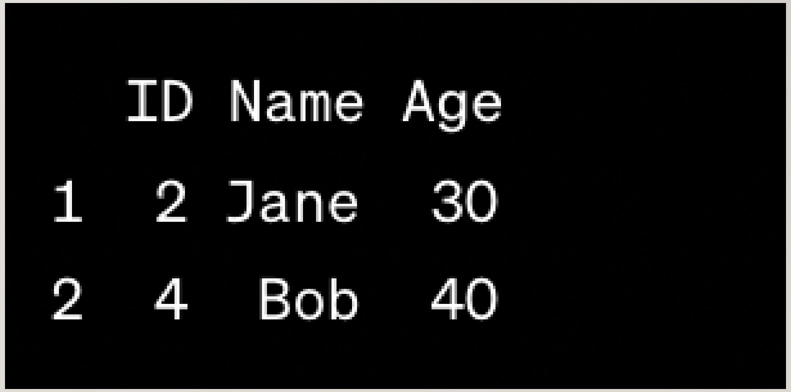
```r
# Example using setdiff()
vector1 <- c(1, 2, 3, 4, 5)
vector2 <- c(4, 5, 6, 7, 8)

setdiff_result <- setdiff(vector1, vector2)
print(setdiff_result)  # Output: 1 2 3
```

SLICE()

- In R, the slice() function is used to extract specific rows from a data frame or tibble based on their positions. It allows you to select rows by specifying the row indices or a range of row indices.

EXAMPLE

- df <- data.frame(ID = 1:5, Name = c("John", "Jane", "Alice", "Bob", "Eve"),Age = c(25, 30, 35, 40, 45))
- # Extract the second and fourth rows
- result <- slice(df, c(2, 4))
- # Print the result
- print(result)



```
  ID Name Age
1  2 Jane  30
2  4  Bob  40
```

INTERSECT() AND UNION()

- In R, the intersect() and union() functions are used to perform set operations on vectors or data frames. Here's an explanation of each function:

1. intersect(x, y): Returns a vector or data frame containing the common elements between two input vectors or data frames x and y. The resulting elements are unique and appear in the order of x. If the input arguments are data frames, the columns must have the same names and types.
2. union(x, y): Returns a vector or data frame containing all the unique elements from both input vectors or data frames x and y. The resulting elements appear in the order they first appear in the combined vectors. If the input arguments are data frames, the columns must have the same names and types.