Learning R programming

If you only know part of the name, such as the first or last name, you can use grepl() in R for partial matching. Here’s how you can achieve this:

**Example Code**

# Sample dataset

data <- data.frame(

Name = c("Alice Smith", "Bob Johnson", "Charlie Brown", "David Clark"),

Age = c(25, 30, 35, 40),

Gender = c("Female", "Male", "Male", "Male")

)

# Part of the name to search

partial\_name <- "Bob" # Can be "Smith", "Johnson", etc.

# Find rows where the name contains the partial string

result <- data[grepl(partial\_name, data$Name, ignore.case = TRUE), ]

# Print the result

print(result)

**Real Example >>>>>>.**

partial\_name <- "pranav"

result <- bamudb[grepl(partial\_name, data$NAME.OF.THE.APPLICANT, ignore.case = TRUE), ]

result

View(result)

**Explanation**

1. **grepl(partial\_name, data$Name, ignore.case = TRUE)**:
   * grepl() searches for the partial\_name in the Name column.
   * ignore.case = TRUE makes the search case-insensitive.
2. **data[...]**:
   * Subsets the dataset to rows where the condition is TRUE.

**Example Output**

If partial\_name is "Bob", the output will be:

markdown

Copy code

Name Age Gender

2 Bob Johnson 30 Male

**Notes**

* If you have first names and last names in separate columns, adjust the logic:

# First and last names in separate columns

data <- data.frame(

FirstName = c("Alice", "Bob", "Charlie", "David"),

LastName = c("Smith", "Johnson", "Brown", "Clark"),

Age = c(25, 30, 35, 40),

Gender = c("Female", "Male", "Male", "Male")

)

# Search by first name or last name

partial\_name <- "Smith"

result <- data[grepl(partial\_name, data$FirstName, ignore.case = TRUE) |

grepl(partial\_name, data$LastName, ignore.case = TRUE), ]

# Print the result

print(result)

**colSums()method:**

The command colSums(is.na(bamudb)) in R is used to calculate the total number of missing (NA) values in each column of the dataset bamudb.

### Explanation:

1. **is.na(bamudb)**:
   * Checks each element of the dataset bamudb to see if it is NA.
   * Returns a logical matrix of the same dimensions as bamudb where TRUE indicates an NA value.
2. **colSums(...)**:
   * Sums up the number of TRUE values (i.e., NAs) in each column of the dataset.

### Example:

# Example dataset

bamudb <- data.frame(

ID = c(1, 2, 3, 4),

Name = c("Alice", NA, "Charlie", "David"),

Age = c(25, 30, NA, 40)

)

# Count missing values in each column

missing\_counts <- colSums(is.na(bamudb))

# Print the result

print(missing\_counts)

### Output:

markdown

Copy code

ID Name Age

0 1 1

### Interpretation:

* The ID column has 0 missing values.
* The Name column has 1 missing value.
* The Age column has 1 missing value.

### Use Case:

This is particularly helpful for quickly identifying columns with missing data in a dataset. You can then decide how to handle the missing values, such as by imputation, removal, or other strategies.

**remove columns where all the values are NA:**

To remove columns where all the values are NA in an R dataset, you can use the following code:

### Code Example

# Example dataset

data <- data.frame(

ID = c(1, 2, 3, 4),

Name = c("Alice", NA, "Charlie", "David"),

Age = c(25, 30, NA, 40),

AllNA = c(NA, NA, NA, NA) # A column with all NA values

)

# Remove columns where all values are NA

data\_cleaned <- data[, colSums(!is.na(data)) > 0]

# Print the result

print(data\_cleaned)

### Explanation:

1. **is.na(data)**:
   * Creates a logical matrix where TRUE represents NA values.
2. **!is.na(data)**:
   * Negates the logical matrix, where TRUE represents non-NA values.
3. **colSums(!is.na(data))**:
   * Sums the number of non-NA values for each column.
4. **colSums(!is.na(data)) > 0**:
   * Returns TRUE for columns with at least one non-NA value.
5. **data[, ...]**:
   * Subsets the dataset, keeping only columns where the condition is TRUE.

### Example Output:

ID Name Age

1 1 Alice 25

2 2 <NA> 30

3 3 Charlie NA

4 4 David 40

### Alternative Using dplyr:

If you prefer using the dplyr package:

library(dplyr)

data\_cleaned <- data %>% select(where(~ any(!is.na(.))))

print(data\_cleaned)

code …

# N===================================== =================================

# ----------- remove those columns where all the values are NA ----------------

# Remove columns where all values are NA

View(colSums(is.na(bamudb)))

data\_cleaned <- bamudb[, colSums(!is.na(bamudb)) == 18736]

# Print the result

print(data\_cleaned)

View(data\_cleaned)

summary(data\_cleaned)

View(summary(data\_cleaned))

nrow(data\_cleaned)

ncol(data\_cleaned)

### NOW CAN SEE THE THERE ARE ONLY 152 COLS ARE THER REMOVED UNWANTED REST OF THE COLUMNS

# ............. EARLIER WE HAD 815 COLUMNS

# N===================================== =====================