

PRACTICAL NO:5

AIM:Sorting data using arrange() in R.

The screenshot shows the RStudio interface with the following components:

- Source:** Contains the R script for loading and inspecting the data.
- Console:** Displays the output of the R script, including the column specification and the first few rows of the data.
- Environment:** Lists the objects in the environment, including the loaded data frames.
- Files:** Shows the file explorer with the location of the data files.

```
R - R4.1.2. ~ /
> # Fixed & robust dplyr script for your uploaded csv
> library(dplyr)
> # file path (your uploaded csv)
> file_path <- "/mnt/data/StudentsPerformance - StudentsPerformance.csv"
> # Load data
> students <- read_csv("My Web Sites/StudentsPerformance - StudentsPerformance.csv")
Rows: 1000 Columns: 8
-- Column specification -----
Delimiter: ","
chr (5): gender, race/ethnicity, parental level of education, lunch, test pr...
dbl (3): math score, reading score, writing score

i use 'spec()' to retrieve the full column specification for this data.
i specify the column types or set 'show_col_types = FALSE' to quiet this message.
> # Helper: normalized column names (lowercase, remove non-alnum)
> norm <- function(x) tolower(gsub("[^a-z0-9]", "", x))
> cols <- colnames(students)
> norm_cols <- setNames(norm(cols), cols) # named vector: original -> normalized
> # Helper to find best match from candidate names
> find_col <- function(candidates) {
+   cand_norm <- sapply(candidates, function(x) tolower(gsub("[^a-z0-9]", "", x)))
+   for (cn in cand_norm) {
+     match_idx <- which(norm_cols == cn)
+     if (length(match_idx) > 0) return(names(norm_cols)[match_idx[1]])
+   }
+   return(NA_character_)
+ }
> # candidate lists for columns (add more variants if you like)
> math_candidates <- c("math.score", "math_score", "math score", "math", "maths", "m
aths.score")
> reading_candidates <- c("reading.score", "reading_score", "reading score", "reading",
"readingscore")
> writing_candidates <- c("writing.score", "writing_score", "writing score", "writing",
"writingscore")
> gender_candidates <- c("gender", "sex")
```

The screenshot shows the RStudio interface with the following components:

- Source:** Contains the R script for sorting the data by math score.
- Console:** Displays the output of the R script, including the sorted data and the first 5 rows.
- Environment:** Lists the objects in the environment, including the sorted data frame.
- Files:** Shows the file explorer with the location of the data files.

```
R - R4.1.2. ~ /
> # Helper: normalized column names (lowercase, remove non-alnum)
> norm <- function(x) tolower(gsub("[^a-z0-9]", "", x))
> cols <- colnames(students)
> norm_cols <- setNames(norm(cols), cols) # named vector: original -> normalized
> # Helper to find best match from candidate names
> find_col <- function(candidates) {
+   cand_norm <- sapply(candidates, function(x) tolower(gsub("[^a-z0-9]", "", x)))
+   for (cn in cand_norm) {
+     match_idx <- which(norm_cols == cn)
+     if (length(match_idx) > 0) return(names(norm_cols)[match_idx[1]])
+   }
+   return(NA_character_)
+ }
> # candidate lists for columns (add more variants if you like)
> math_candidates <- c("math.score", "math_score", "math score", "math", "maths", "m
aths.score")
> reading_candidates <- c("reading.score", "reading_score", "reading score", "reading",
"readingscore")
> writing_candidates <- c("writing.score", "writing_score", "writing score", "writing",
"writingscore")
> gender_candidates <- c("gender", "sex")
> # Detect actual column names
> math_col <- find_col(math_candidates)
> reading_col <- find_col(reading_candidates)
> writing_col <- find_col(writing_candidates)
> gender_col <- find_col(gender_candidates)
> # If required columns not found, print available columns and stop
> required <- c(math_col, reading_col, writing_col)
> if (any(is.na(required))) {
+   message("Could not find all required score columns. Detected columns in the file ar
e:\n")
+   print(cols)
+   stop("Missing one or more of math/reading/writing columns. update the candidate lis
ts if your column names are different.")
+ }
> # Example 1: Sort by math score ascending
> students_sorted_math <- students %>%
+   arrange(.data[math_col])
> cat("First 5 rows sorted by", math_col, "(ascending):\n")
First 5 rows sorted by math score (ascending):
> print(head(students_sorted_math, 5))
# A tibble: 5 x 8
  gender race/ethnicity parental level of educa-1 lunch test preparation cou-2
<chr> <chr> <chr> <chr> <chr>
1 female group C some high school free- none
2 female group B high school free- none
3 female group B some high school free- none
4 female group B some college stan- none
5 female group C some college free- none
# i abbreviated names: 1: parental level of education ,
# 2: test preparation course"
# i 3 more variables: 'math score' <dbl>, 'reading score' <dbl>,
```

SHETH L.U.J. AND SIR M.V. COLLEGE

RStudio interface showing R code for sorting data. The console output displays the first 5 rows sorted by reading score (descending) and the first 10 rows sorted by gender and then writing score (descending).

```
# Example 2: Sort by reading score descending
> students_sorted_reading_desc <- students %>%
+   arrange(desc(data[[reading_col]]))
> cat("\nFirst 5 rows sorted by", reading_col, "(descending):\n")

First 5 rows sorted by reading score (descending):
> print(head(students_sorted_reading_desc, 5))
# A tibble: 5 x 8
  gender 'race/ethnicity' parental level of educa-1 lunch test preparation cou-2
  <chr>   <chr>           <chr>           <chr>   <chr>
1 female group D      master's degree    stan~ none
2 female group E      bachelor's degree  stan~ completed
3 male group E        associate's degree free~ completed
4 female group C      bachelor's degree  stan~ completed
5 female group D      some high school  stan~ completed

# 1 abbreviated names: 1: 'parental level of education',
# 2: 'test preparation course'
# 3 more variables: 'math score' <dbl>, 'reading score' <dbl>,
# 'writing score' <dbl>

# Example 3: Sort by gender (if found) then by writing score descending
> if (!is.na(gender_col)) {
+   students_sorted_gender_writing <- students %>%
+     arrange(data[[gender_col]], desc(data[[writing_col]]))
+   cat("\nFirst 10 rows sorted by", gender_col, "and then", writing_col, "(desc):\n")
+   print(head(students_sorted_gender_writing, 10))
+ } else {
+   cat("\nGender column not detected; skipping gender+writing sort example.\n")
+ }
```

Environment pane shows loaded objects: high_math_sorted (50 obs. of 8 variables), high_price_subset (8 obs. of 13 variables), housing (545 obs. of 13 variables), insurance (545 obs. of 13 variables), large_highprice_s... (4 obs. of 13 variables), mainroad_baseament (168 obs. of 13 variables), manv_bathrooms (1 obs. of 13 variables).

RStudio interface showing R code for filtering data. The console output displays the top 5 high math scorers (> 90) with lowest writing scores.

```
# Example 4: Filter (high math > 90) then arrange by writing score ascending
> # (adjust threshold 90 if your scoring range differs)
> high_math_sorted <- students %>%
+   filter(data[[math_col]] > 90) %>%
+   arrange(data[[writing_col]])
> cat("\nTop 5 high math scorers (> 90, ", " with lowest writing scores:\n")

Top 5 high math scorers (> 90) with lowest writing scores:
> print(head(high_math_sorted %>% select(all_of(c(gender_col, math_col, writing_col))),
5))
# A tibble: 5 x 3
  gender 'math score' 'writing score'
  <chr>   <dbl>       <dbl>
1 male    94         71
2 male    91         76
3 male    94         78
4 male    92         78
5 male    91         79
```

Environment pane shows loaded objects: high_math_sorted (50 obs. of 8 variables), high_price_subset (8 obs. of 13 variables), housing (545 obs. of 13 variables), insurance (545 obs. of 13 variables), large_highprice_s... (4 obs. of 13 variables), mainroad_baseament (168 obs. of 13 variables), manv_bathrooms (1 obs. of 13 variables).