Automating Geothermal Data Processing with R

Veronika Slavíková

2025-08-25

Introduction

This project details a data processing workflow developed in R for a geothermal laboratory. The script automates the updating of geothermal data stored in an Excel spreadsheet by calculating new values from raw .dat files. I created the code in posit.cloud, a cloud-based environment for data analysis, to ensure a reproducible and accessible process. The project demonstrates my ability to work with various data formats, perform data manipulation, and develop a robust, scalable data pipeline.

Key Skills Demonstrated

- Data Cleaning and Preprocessing: Reading and cleaning data from multiple file types (.xlsx, .dat), renaming columns, and converting data types.
- Data Manipulation with R: Using the tidyverse suite of packages to transform and analyze data, including file listing, string splitting, and data frame manipulation.
- Automation and Scripting: Developing a reusable R script to automate a repetitive data processing task, improving efficiency and reducing manual effort.
- File I/O and Management: Importing data from specified file paths and exporting updated data to a new file, ensuring data integrity.
- Problem-Solving: Identifying and addressing data formatting requirements (e.g., specific filename conventions) to ensure the script functions correctly.
- Structured Programming: Writing a modular script with clear steps for data loading, processing, and exporting, including the use of functions to avoid code duplication.

Data Source

The datasets used in this project are sample files with invalid geothermal data. The primary purpose of this document is to demonstrate the calculation and data processing methodology, which will be applied to real datasets in future projects.

Data loading and processing

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(readxl)
library(writexl)
library(tools)
library(data.table)
```

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
# Set the path to your Excel file and read the data
setwd("/cloud/project/Katarina_data")
data1 <- read_excel(path = "HG_vrt.xlsx", range = "AB2:AH360")
```

```
## New names:
## * `TC` -> `TC...4`
## * `TD` -> `TD...5`
## * `TC` -> `TC...6`
## * `TD` -> `TD...7`
```

```
# Convert Metraz to numeric and round down, and rename columns
data1$Metraz <- floor(as.numeric(data1$Metraz))
```

```
## Warning: NAs introduced by coercion
```

```
data1 <- data1 %>% rename(TC_1 = TC...4, TD_1 = TD...5, TC_2 = TC...6, TD_2 = TD...7)

# --- Find all DAT files in the working directory ---
# This line lists all files ending with ".dat"
dat_files <- list.files(path = ".", pattern = "\\.dat$", full.names = FALSE)
```

Extracting depth data from DAT file names, selecting the first six columns from DAT files, calculating the averages of values for TC and TD, and writing them to the "data1" dataframe.

```
# Check if any DAT files were found
if (length(dat_files) == 0) {
  cat("No DAT files found in the working directory.\n")
} else {
  # --- Loop through each DAT file ---
  for (input_file in dat_files) {
    cat(paste("Processing file:", input_file, "\n"))
    # Extract numbers from filename
    filename_no_ext <- tools::file_path_sans_ext(input_file)
    parts <- strsplit(filename_no_ext, "_")
    first_number <- as.double(parts[[1]][1])
    second_number <- as.double(parts[[1]][2])

    # Read the data from the current DAT file and select the first 6 columns
    dat_data <- read.table(input_file, header = FALSE, sep = "", stringsAsFactors = FALSE)
    selected_data <- dat_data[, 1:6]

    # --- Function to calculate average and update data frame ---
    # This function now takes the target column names as arguments
    update_data <- function(row_indices, col_index_A, col_index_F, metraz_val, col_name_tc, col_name_td
      # Calculate the average for 'TC' (column 1 in selected_data)
      avg_tc <- (selected_data[row_indices[1], col_index_A] + selected_data[row_indices[2], col_index_A

      # Calculate the average for 'TD' (column 6 in selected_data)
      avg_td <- (selected_data[row_indices[1], col_index_F] + selected_data[row_indices[2], col_index_F

      # Find the row in data1 to update
      row_to_update <- which(data1$Metraz == metraz_val)

      # Update the data1 data frame with the calculated averages
      data1[row_to_update, col_name_tc] <<- avg_tc
      data1[row_to_update, col_name_td] <<- avg_td

      # Return the updated data frame (though we are using <<- to modify it directly)
      return(data1)
    }

    # --- Apply the function for the first number (first_number) ---
    data1 <- update_data(c(1, 3), 1, 6, first_number, "TC_1", "TD_1") # For TC_1 and TD_1
    data1 <- update_data(c(5, 7), 1, 6, first_number, "TC_2", "TD_2") # For TC_2 and TD_2

    # --- Apply the function for the second number (second_number) ---
    data1 <- update_data(c(2, 4), 1, 6, second_number, "TC_1", "TD_1") # For TC_1 and TD_1
    data1 <- update_data(c(6, 8), 1, 6, second_number, "TC_2", "TD_2") # For TC_2 and TD_2
  }
}
```

```
## Processing file: 188_223.dat
## Processing file: 230_199.dat
## Processing file: 312_161.dat
```

Export the added columns TC and TD to a new XLSX file named "HG_vrt_updated.xlsx"

```
# --- Final Step: Write the updated data frame to a new Excel file ---
# We'll save the modified data frame back to a new Excel file using writexl.
# It's good practice to save to a new file to avoid issues with overwriting
# the original source file.
output_data1_file <- "HG_vrt_updated.xlsx"
writexl::write_xlsx(data1, output_data1_file)
```

```r
# Print a confirmation message
cat(paste("Successfully inserted averages and saved updated data to", output_data1_file, "\n"))
```

```
## Successfully inserted averages and saved updated data to HG_vrt_updated.xlsx
```