Set Names to Lowercase for Multiple Dataframes in R

A practical approach to batch-renaming column headers across every dataframe in your workspace

Ronald 'Ryy' G. Thomas

2026-04-30

Table of contents

Figure 1: Index cards being reorganised on a warm wooden desk.

Batch renaming: a small automation that compounds over time.

1 Introduction

Reliably lowercasing the column names of every dataframe in an R session becomes necessary when working with messy consulting datasets where half the tables use `SCREAMING_CASE` and the other half use `Title_Case`. Manually fixing each one is tedious, and a programmatic solution is clearly needed.

The core challenge is straightforward: given a workspace that contains a mix of dataframes, vectors, and other objects, how does one (a) identify which objects are dataframes, and (b) convert all of their column names to lowercase in one pass? The task sounds simple, but there are a few subtleties, especially when tibbles and other multi-class objects enter the picture.

This post walks through a reliable solution using base R functions like `eapply()` and `list2env()` alongside an optional `purrr`-based alternative, and it should take about ten minutes to work through from start to finish.

1.1 Motivations

The following considerations motivated this exploration:

- Time spent manually renaming columns every time a new dataset arrives from a collaborator who uses uppercase headers adds up quickly.
- Inconsistent column name casing causes silent failures in `dplyr` joins; `merge(df1, df2, by = "id")` does not match `ID` to `id` without explicit intervention.

- A solution that can be dropped into any project without installing extra packages has broad applicability.
- Batch operations over workspace objects represent an underappreciated area of R programming.
- Understanding how R environments work at a deeper level is a worthwhile investment for daily work.

1.2 Objectives

1. Identify all dataframe objects in the current R environment programmatically.
2. Convert their column names to lowercase in a single batch operation.
3. Write the modified dataframes back to the global environment without manual assignment.
4. Compare a base R approach (`eapply` + `lapply`) with a `purrr`-based functional alternative.

Errors and better approaches are welcome; see the Feedback section at the end.

Figure 2: A workspace ready for data wrangling.

2 Prerequisites and Setup

To follow along you need R (version 4.1 or later for the native pipe `|>`) and optionally the `purrr` package for the functional alternative. No other dependencies are required.

```
library(purrr)
library(tibble)
```

Background. This tutorial assumes familiarity with basic R data structures (vectors, lists, dataframes) and the idea that R stores objects in environments. If you have used `ls()` to list your workspace objects, you have enough background to follow along.

3 What is Column Name Standardisation?

Column name standardisation is the practice of enforcing a single naming convention across all columns in a dataset. Think of it like enforcing a dress code: when every column follows the same convention, downstream code does not need to guess whether a column is called `Age`, `AGE`, or `age`.

The most common convention in the R ecosystem is `snake_case` (all lowercase, words separated by underscores). The `janitor::clean_names()` function is the gold standard for comprehensive cleaning, but for simple lowercasing of existing names the base R function `tolower()` is sufficient.

The challenge addressed here is not cleaning a single dataframe (that is a one-liner). The challenge is doing it for every dataframe in a workspace that may contain dozens of objects of different types.

4 Getting Started: Setting Up Test Data

We begin by constructing a small workspace with a mix of object types. Two objects are dataframes (one a plain `data.frame`, one a `tibble`), and two are simple vectors.

```
rm(list = ls())

aa <- data.frame(COL_1 = letters[6:4], COL_2 = 1:3)
bb <- tibble(COL_1 = letters[6:8], COL_2 = 1:3)
cc <- 1:10
dd <- letters[1:4]
```

A quick look at the workspace confirms four objects:

```
ls()
```

```
[1] "aa" "bb" "cc" "dd"
```

And we can verify the column names are uppercase:

```
names(aa)
```

```
[1] "COL_1" "COL_2"
```

```
names(bb)
```

```
[1] "COL_1" "COL_2"
```

The goal is to convert `COL_1` and `COL_2` to `col_1` and `col_2` in both `aa` and `bb`, without touching `cc` or `dd`.

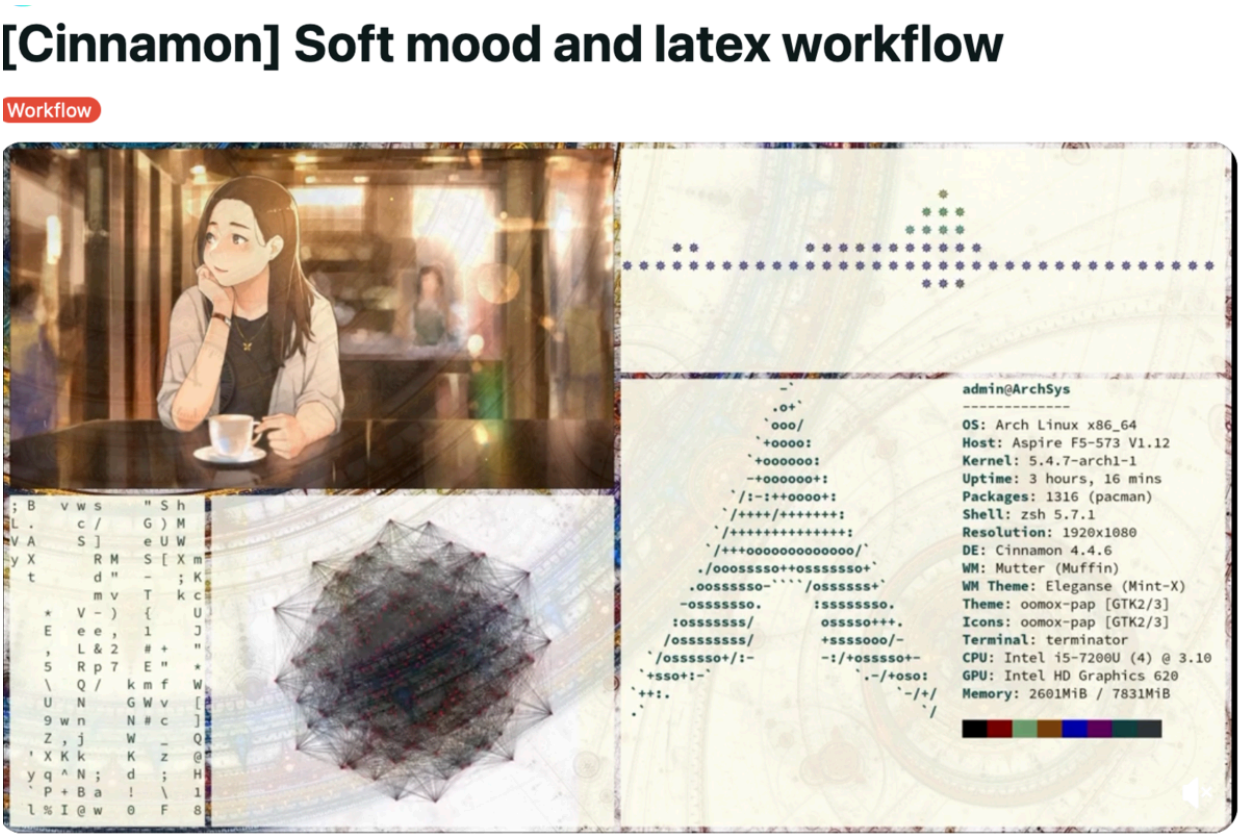

Figure 3: Soft tones for a moment of reflection

A brief pause before we dig into the solution.

5 Building the Solution

The approach has three steps: (1) find all dataframes in the environment, (2) lowercase their column names, and (3) write them back to the global environment.

5.1 Step 1: Find All Dataframes

The function `eapply()` applies a function to every object in an environment; it is the environment-level analogue of `lapply()`. It is used here to test each object and return it only if it passes `is.data.frame()`:

```r
df0 <- eapply(.GlobalEnv, \(x) {
  if (is.data.frame(x)) return(x)
})
df1 <- df0[!sapply(df0, is.null)]
names(df1)
```

```
[1] "bb" "aa"
```

The `eapply()` call returns a named list. Objects that are not dataframes come back as `NULL`, so we filter them out. The result, `df1`, is a named list containing only the dataframes.

Note that `is.data.frame()` returns `TRUE` for tibbles as well, because tibbles inherit from `data.frame`. This means the approach works even when your workspace contains a mix of plain dataframes and tibbles.

5.2 Step 2: Lowercase the Column Names

With the dataframes collected in a list, a single `lapply()` call converts every column name to lowercase:

```r
df2 <- lapply(df1, \(x) {
  names(x) <- tolower(names(x))
  x
})
```

Each element of `df2` is now a dataframe with lowercase column names. The original objects in the global environment remain unchanged until we complete the final step.

5.3 Step 3: Write Back to the Global Environment

The function `list2env()` takes a named list and assigns each element to the specified environment, using the list names as object names:

```r
list2env(df2, .GlobalEnv)
```

```
<environment: R_GlobalEnv>
```

Because `df1` preserved the original object names (`aa` and `bb`), `list2env()` overwrites those objects in the global environment with their lowercase-named versions.

5.4 Alternative Approach

An alternative one-liner extracts just the names of the dataframes, which you can then iterate over separately. This approach comes from a Tutorials Point article on listing dataframes:

```
  df_names <- names(
    which(unlist(eapply(.GlobalEnv, is.data.frame)))
  )
  df_names
```

```
[1] "bb" "aa"
```

You can also write the entire pipeline using `purrr` for a more functional style:

```
  .GlobalEnv |>
    as.list() |>
    keep(is.data.frame) |>
    map(\(x) {
      names(x) <- tolower(names(x))
      x
    }) |>
    list2env(.GlobalEnv)
```

The `purrr` version uses `keep()` to filter for dataframes and `map()` to apply the renaming function, which some readers may find more readable than the `eapply()` + `sapply()` chain.

6 Checking Our Work

Let us verify that the column names have been lowercased in the global environment:

```
  names(aa)
```

```
[1] "col_1" "col_2"
```

```
  names(bb)
```

```
[1] "col_1" "col_2"
```

And confirm that the non-dataframe objects are untouched:

```
  cc
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
  dd
```

```
[1] "a" "b" "c" "d"
```

The dataframes now have lowercase column names, and the vectors remain exactly as they were.

6.1 Things to Watch Out For

1. **The variable name trap.** Early drafts of this code often use inconsistent variable names between the `eapply()` output and the null-filter step. Ensure both lines reference the same object.
2. **Tibbles are dataframes.** `is.data.frame()` returns `TRUE` for tibbles, which is usually what you want. If you need to exclude tibbles, test with `inherits(x, "tbl_df")`.
3. **`list2env()` overwrites silently.** There is no confirmation prompt. If you have an object named `aa` that is not a dataframe, but your list also contains an element named `aa`, it will be overwritten. Always inspect `names(df1)` before calling `list2env()`.
4. **Column names with special characters.** `tolower()` handles ASCII letters but does not address spaces, dots, or other non-standard characters. For thorough cleaning, consider `janitor::clean_names()` instead of `tolower()`.
5. **Side effects in the global environment.** This approach mutates the global environment in place. In production code, prefer passing dataframes explicitly rather than relying on environment-level side effects.

{.img-fluid fig-alt="UCSD Geisel Library building against a clear sky, providing a visual break before the concluding sections"}

Taking a moment to step back before summarising what we have learnt.

7 What Did We Learn?

7.1 Lessons Learnt

Conceptual Understanding:

- R environments are first-class objects that can be iterated over, queried, and modified programmatically.
- `eapply()` is the environment analogue of `lapply()` and is underappreciated in everyday R programming.
- `list2env()` bridges the gap between list-based processing and environment-based storage, enabling batch updates.
- Column name standardisation is a defensive practice that prevents subtle join and filter bugs downstream.

Technical Skills:

- Combining `eapply()` with `is.data.frame()` provides a reliable way to survey the contents of any environment.
- Anonymous functions (`\(x) ...`) keep one-off logic inline without polluting the namespace.
- The `purrr::keep()` + `purrr::map()` chain offers a readable alternative to the base R approach.
- `tolower()` is sufficient for simple case conversion; `janitor::clean_names()` handles edge cases (dots, spaces, duplicates).

Gotchas and Pitfalls:

- Variable name inconsistencies between pipeline steps are the most common source of bugs in this pattern.
- `list2env()` is a side-effect function; use it deliberately and inspect the list before calling it.
- Tibbles pass `is.data.frame()` checks, which is correct behaviour but can surprise newcomers.
- This pattern does not recurse into nested environments or package namespaces; it operates only on the environment you pass to `eapply()`.

7.2 Limitations

- The approach operates only on the global environment by default. Adapting it for nested environments or package namespaces requires additional logic.
- `tolower()` handles ASCII characters only. Column names containing accented characters or Unicode may need `stringi::stri_trans_tolower()` for correct conversion.
- There is no undo mechanism. Once `list2env()` runs, the original uppercase names are gone unless you saved a backup.
- The pattern does not handle naming collisions; if lowercasing creates duplicate column names (e.g., `Name` and `name` both become `name`), one will silently overwrite the other.
- Performance has not been tested on workspaces with hundreds of large dataframes. For such cases, a targeted approach using explicit object names may be more efficient.

7.3 Opportunities for Improvement

1. **Wrap the logic in a reusable function** that accepts an environment argument and returns a summary of which objects were modified.
2. **Add a dry-run mode** that reports which dataframes would be affected and what their new column names would be, without actually modifying anything.
3. **Integrate with `janitor::clean_names()`** for more thorough standardisation (handling dots, spaces, duplicates, and non-ASCII characters).
4. **Add logging** so that each rename operation is recorded, making it easier to audit changes in a collaborative workflow.
5. **Extend to other object types** such as matrices or `data.table` objects, which have their own column naming conventions.
6. **Write unit tests** using `testthat` to verify the function works correctly with edge cases (empty dataframes, dataframes with no columns, single-column dataframes).

8 Wrapping Up

This post walked through a practical pattern for finding every dataframe in an R workspace and lowercasing its column names in a single batch operation. The core mechanism (`eapply()` to survey the environment, `lapply()` to transform, and `list2env()` to write back) is simple, requires no external packages, and works with both plain dataframes and tibbles.

The most valuable outcome is learning about `eapply()`. It is one of those base R functions that rarely appears in tutorials but turns out to be exactly the right tool when operating on an entire environment programmatically. Understanding how R environments work at this level builds confidence in writing code that manages multiple datasets.

If you are working with messy, multi-source data and find yourself manually renaming columns, give this pattern a try. The main takeaways are:

- `eapply(.GlobalEnv, is.data.frame)` identifies all dataframes in the workspace.
- `lapply()` with `tolower(names(x))` lowercases column names across all dataframes at once.
- `list2env()` writes the modified dataframes back to the global environment.
- The `purrr` alternative (`keep()` + `map()`) provides a tidyverse-flavoured version of the same logic.

9 See Also

Key resources:

- Advanced R: Environments: Hadley Wickham's authoritative treatment of R environments.
- purrr documentation: Official reference for functional programming in R.
- janitor::clean_names(): The gold standard for column name standardisation.
- Tutorials Point: List dataframes in R: Source of the alternative one-liner approach.

10 Reproducibility

This post was written with:

```
sessionInfo()
```

```
R version 4.5.3 (2026-03-11)
Platform: aarch64-apple-darwin20
Running under: macOS Tahoe 26.4.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] tibble_3.3.1 purrr_1.2.1

loaded via a namespace (and not attached):
 [1] digest_0.6.39   fastmap_1.2.0   xfun_0.57       magrittr_2.0.5
 [5] glue_1.8.0      knitr_1.51      parallel_4.5.3  pkgconfig_2.0.3
 [9] htmltools_0.5.9 rmarkdown_2.31  lifecycle_1.0.5 cli_3.6.5
[13] vctrs_0.7.2     compiler_4.5.3  tools_4.5.3     pillar_1.11.1
[17] evaluate_1.0.5  yaml_2.3.12     otel_0.2.0      rlang_1.1.7
[21] jsonlite_2.0.0
```

Packages used: `purrr`, `tibble` (optional; base R suffices for the core approach).

To reproduce: Copy the code blocks into an R session running R 4.1 or later. No external data files are required; the tutorial uses inline test data.

11 Let's Connect

Have questions, suggestions, or spot an error? Let me know.

- **GitHub:** rgt47
- **Twitter/X:** @rgt47
- **LinkedIn:** Ronald Glenn Thomas

- **Email:** Contact form

I would enjoy hearing from you if:

- You spot an error or a better approach to any of the code in this post.
- You have suggestions for topics you would like to see covered.
- You want to discuss R programming, data science, or reproducible research.
- You have questions about anything in this tutorial.
- You just want to say hello and connect.