Package: TidyPanel (via r-universe)

May 11, 2026

Title Universal Messy Panel Data Cleaner

Version 0.1.2

Description A robust toolkit designed to standardize and clean complex tabular data from commercial enterprise systems, healthcare records, logistics software, and HR databases. Features include intelligent regex parsing for domain-specific noise (currencies, percentages), gap-based block clustering, and automated messy table resolution. Methods draw on tidy data principles described in Wickham (2014) <doi:10.18637/jss.v059.i10> and the 'readxl' parsing infrastructure described in Wickham & Bryan (2023) <https://readxl.tidyverse.org>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports dplyr, stringr, readxl

Suggests knitr, rmarkdown, testthat, writexl

VignetteBuilder knitr

NeedsCompilation no

Author Tony Lu [aut, cre]

Maintainer Tony Lu <xulunt123@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-05-11 21:45:29 UTC

RemoteUrl https://github.com/cran/TidyPanel

RemoteRef HEAD

RemoteSha 9c732603bb6061579fa695febc0e3f51659e942e

Contents

`clean_variable_names` *Standardize and Clean Variable Names*

Description

`clean_variable_names()` standardizes column names in a messy data frame. It converts all names to snake_case, strips special characters (except _), translates Excel serial dates (e.g., `44197`) into ISO date strings (`2021-01-01`), and maps common financial/academic synonyms (e.g., `gvkey`, `permno`, `cusip`) to standard names (`id`, `ticker`).

Usage

```
clean_variable_names(data)
```

Arguments

`data`	A `data.frame`. The data frame with messy column names.

Value

A `data.frame` with the same data but standardized column names.

Examples

```
# Toy example: standardize column names in a data frame
df <- data.frame(
  `Total Revenue ($)` = 1,
  `PERMNO` = 3,
  `My Custom Column!` = 4,
  check.names = FALSE
)
clean_df <- clean_variable_names(df)
colnames(clean_df)
# Returns: c("revenue", "id", "my_custom_column")

# Excel serial dates are also handled
df2 <- data.frame(`44197` = 2, check.names = FALSE)
colnames(clean_variable_names(df2))
# Returns: "2021-01-01"
```

infer_data_types *Smart Type Coercion & NA Recognition*

Description

infer_data_types() scans character columns in a data frame, identifies common financial placeholders for missing data (e.g., "-", "N/A", "n.m."), safely replaces them with NA, and then coerces the column to numeric or Date if a high percentage of the remaining values match those types.

Usage

```
infer_data_types(
  data,
  na_strings = c("-", "N/A", "n/a", "n.m.", "n.m", "NA", "null", "NULL", "."),
  num_threshold = 0.95
)
```

Arguments

data	A data.frame.
na_strings	A character vector of strings to be interpreted as NA.
num_threshold	Numeric between 0 and 1. The proportion of valid numbers required to convert a column to numeric. Default is 0.95.

Value

A data.frame with inferred data types.

Examples

```
# Clean financial placeholders and coerce to numeric
df <- data.frame(val = c("1.5", "-", "2.0", "N/A"), stringsAsFactors = FALSE)
df_clean <- infer_data_types(df)
df_clean$val  # numeric: c(1.5, NA, 2.0, NA)
is.numeric(df_clean$val)  # TRUE
```

normalize_units *Normalize Numeric Columns Based on Header Unit Declarations*

Description

normalize_units() scans the column names of a data frame for financial/scientific unit declarations (e.g., "Revenue (in millions)", "Assets ($k)", "Employees ('000)"). It automatically multiplies the numeric values in the corresponding columns by the detected multiplier (1,000, 1,000,000, etc.) and optionally strips the unit declaration from the column name.

Usage

```
normalize_units(data, strip_units = TRUE)
```

Arguments

data	A data.frame. The data frame to be normalized.
strip_units	Logical. If TRUE, removes the unit declarations from the column names. Default is TRUE.

Value

A data.frame with the normalized data and updated column names.

Examples

```
# Scale columns declared in millions and thousands
df <- data.frame(
  `Revenue ($M)` = c(1.5, 2.0),
  `Cost (in thousands)` = c(500, 600),
  check.names = FALSE
)
result <- normalize_units(df)
result$Revenue  # c(1500000, 2000000)
result$Cost     # c(500000, 600000)
```

---

read_messy_panel *Robust Parsing and Extraction of Messy Excel Panel Data*

Description

read_messy_panel() is an industrial-grade parser designed to extract clean, standardized data frames from heavily malformed, human-readable Excel reports (e.g., financial statements, ERP exports). It automatically bypasses decoy rows, stitches N-dimensional hierarchical headers, extracts structural indentation hierarchies (parent-child relationships), amputates embedded subtotals, and standardizes financial/scientific numbers.

Usage

```
read_messy_panel(
  file_path,
  sheet = NULL,
 na_strings = c("", "NA", "#N/A", "NULL", "S", "D", "ND", "N/A", "*", "**", "***", ".",
   "x", "c", "s", "z", "#VALUE!", "#REF!", "#DIV/0!", "#NUM!", "#NAME?", "none", "NR",
    "--", "---", "n.a.", "N.A.", "n/a", "Not Applicable"),
  clean_vars = TRUE,
  auto_pivot = FALSE,
  return_audit = FALSE
)
```

Arguments

file_path	Character string. Path to the Excel file.
sheet	Optional sheet name or index. If NULL (the default), it auto-discovers the first valid data panel across all sheets. If "ALL", parses and merges all sheets.
na_strings	Character vector. Strings to interpret as missing values. Supports complex missing-value lexicons.
clean_vars	Logical. If TRUE (default), standardizes variable names to snake_case using clean_variable_names().
auto_pivot	Logical. If TRUE, attempts to reshape wide temporal columns (e.g., FY2021, Q1_2022) into a long format (time_period, value).
return_audit	Logical. If TRUE, returns a list containing $data (the cleaned data frame) and $audit (a detailed log of all algorithmic modifications made).

Value

If return_audit = FALSE, a cleaned and standardized data.frame. If return_audit = TRUE, a named list containing:

data	The cleaned data.frame.
audit	A data.frame detailing exactly what transformations, truncations, or imputations were applied.

Examples

```
# Toy example: create a small in-memory Excel file and parse it
tmp <- tempfile(fileext = ".xlsx")
df_raw <- data.frame(
  Category = c("Revenue", "Cost", "Total"),
  `2022` = c("1.2M", "800k", "2.0M"),
  `2023` = c("1.5M", "900k", "2.4M"),
  check.names = FALSE
)
writexl::write_xlsx(df_raw, tmp)
result <- read_messy_panel(tmp, auto_pivot = TRUE)
head(result)
unlink(tmp)
```

Index