Package ‘zztable1nextgen’

August 25, 2025

Version 0.1.0

Title Next Generation Summary Tables

Description
A package to create publication-quality summary tables with a flexible and powerful interface.

Author Your Name `<you@example.com>`

License MIT

Encoding UTF-8

RoxygenNote 7.3.2

Contents

`analyze_dimensions_optimized`
Optimized Table Dimension Analysis

Description

Completely redesigned dimension analysis using modern R patterns. Much more efficient and maintainable than the original approach.

Usage

```
analyze_dimensions_optimized(
  x_vars,
  grp_var,
  data,
  strata = NULL,
  missing = FALSE,
  size = FALSE,
  totals = FALSE,
  pvalue = TRUE,
  layout = "console",
  footnotes = NULL
)
```

Arguments

`x_vars`	Character vector of analysis variables
`grp_var`	Character string naming grouping variable
`data`	Data frame containing variables
`strata`	Optional stratification variable name
`missing`	Logical indicating whether to show missing counts
`size`	Logical indicating whether to show group sizes
`totals`	Logical indicating whether to include totals column
`pvalue`	Logical indicating whether to include p-values
`layout`	Character string specifying output format
`footnotes`	Optional footnote specifications

Value

Optimized dimension analysis structure

`apply_theme` *Apply Theme to Blueprint*

Description

Apply Theme to Blueprint

Usage

```
apply_theme(blueprint, theme)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration list

Value

Modified blueprint with theme applied

`apply_theme_to_cell` *Apply Theme to Single Cell*

Description

Apply Theme to Single Cell

Usage

```
apply_theme_to_cell(cell, theme)
```

Arguments

`cell`	Cell object
`theme`	Theme configuration

Value

Themed cell

`apply_theme_to_cells` *Apply Theme to All Cells*

Description

Apply Theme to All Cells

Usage

```
apply_theme_to_cells(blueprint, theme)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration

`blueprint_memory_info` *Get Blueprint Memory Usage Information*

Description

Returns detailed memory usage statistics for blueprint objects, useful for performance monitoring and optimization.

Usage

```
blueprint_memory_info(x, unit = "KB")
```

Arguments

`x`	A table1_blueprint object
`unit`	Character string specifying size unit ("B", "KB", "MB")

Value

Named list with memory usage statistics

`calculate_frequency_table` *Calculate Frequency Table*

Description

Calculate Frequency Table

Usage

```
calculate_frequency_table(x, sort_by = "frequency")
```

Arguments

`x`	Vector (factor, character, or logical)
`sort_by`	Sort by "frequency", "alphabetical", or "none" (default "frequency")

Value

Data frame with levels, counts, and proportions

`calculate_summary_stats` *Calculate Summary Statistics*

Description

Calculate Summary Statistics

Usage

```
calculate_summary_stats(x, na.rm = TRUE)
```

Arguments

`x`	Numeric vector
`na.rm`	Logical, remove NA values (default TRUE)

Value

Named list with summary statistics

`Cell` *Create Cell Object with Validation (User Interface)*

Description

User-facing constructor that provides comprehensive input validation and creates properly validated Cell objects.

Usage

```
Cell(
  type,
  content = NULL,
  data_subset = NULL,
  computation = NULL,
  dependencies = NULL,
  format = list(),
  cached_result = NULL,
  footnote_number = NULL,
  footnote_text = NULL
)
```

Arguments

`type`	Character string specifying cell type. Must be one of: "static", "computation", "separator", "footnote", "footnote_separator"
`content`	Static text content (required for "static" type)
`data_subset`	R expression for data subsetting (required for "computation")
`computation`	R expression for calculation (required for "computation")
`dependencies`	Character vector of variable dependencies
`format`	Named list of formatting options
`cached_result`	Cached computation result (for performance)
`footnote_number`	Integer footnote number (for footnote cells)
`footnote_text`	Character footnote text (for footnote cells)

Value

Validated Cell object of class "cell"

Examples

```
# Static cell
Cell(type = "static", content = "Age (years)")

# Computation cell
Cell(type = "computation",
     data_subset = expression(data$age[data$group == "Treatment"]),
     computation = expression(paste0(round(mean(x), 1), " ± ", round(sd(x), 1))),
     dependencies = c("data", "age", "group"))

# Footnote cell
Cell(type = "footnote", footnote_number = 1,
     footnote_text = "Age measured at baseline")
```

`clear_cell_cache` *Clear Cell Cache*

Description

Clear Cell Cache

Usage

```
clear_cell_cache(cell)
```

Arguments

`cell`	Cell object

`combine_row_content` *Combine Row Content*

Description

Combine Row Content

Usage

```
combine_row_content(row_content, format, theme)
```

Arguments

`row_content`	Character vector of cell contents
`format`	Output format
`theme`	Theme configuration

Value

Single character string for the row

`create_cell_key` *Create Hash Key for Cell Position*

Description

Create Hash Key for Cell Position

Usage

```
create_cell_key(row, col)
```

Arguments

`row`	Integer row number
`col`	Integer column number

Value

Character hash key

`detect_variable_type` *Detect Variable Type*

Description

Detect Variable Type

Usage

```
detect_variable_type(x)
```

Arguments

`x`	Vector to analyze

Value

Character string describing variable type

`escape_html` *Escape HTML Special Characters*

Description

Escape HTML Special Characters

Usage

```
escape_html(text)
```

Arguments

`text`	Character string

Value

HTML-escaped string

`escape_latex` *Escape LaTeX Special Characters*

Description

Escape LaTeX Special Characters

Usage

```
escape_latex(text)
```

Arguments

`text`	Character string

Value

LaTeX-escaped string

`evaluate_cells_vectorized` *Vectorized Cell Evaluation*

Description

Vectorized Cell Evaluation

Usage

```
evaluate_cells_vectorized(cells, data, parallel = FALSE)
```

Arguments

`cells`	List of cell objects
`data`	Data frame
`parallel`	Logical, use parallel processing

Value

List of evaluated results

`evaluate_cell_optimized` *Enhanced Cell Evaluation*

Description

Optimized cell evaluation with better error handling, caching, and performance monitoring.

Usage

```
evaluate_cell_optimized(cell, data, cache_results = TRUE)

evaluate_cell_optimized(cell, data, cache_results = TRUE)
```

Arguments

`cell`	Cell object to evaluate
`data`	Data frame context
`cache_results`	Logical, whether to cache results (default TRUE)
`env`	Evaluation environment
`force_recalc`	Logical to force cache invalidation

Value

Evaluated cell result

Evaluated content or error marker

`evaluate_pvalue_cell` *Evaluate P-value Cell*

Description

Evaluate P-value Cell

Usage

```
evaluate_pvalue_cell(cell, data)
```

Arguments

`cell`	P-value cell
`data`	Data frame

Value

P-value result

`evaluate_summary_cell` *Evaluate Summary Cell*

Description

Evaluate Summary Cell

Usage

```
evaluate_summary_cell(cell, data)
```

Arguments

`cell`	Summary cell
`data`	Data frame

Value

Summary result

`format_cell_content` *Format Cell Content with Theme*

Description

Format Cell Content with Theme

Usage

```
format_cell_content(content, theme, cell_type = "static")
```

Arguments

`content`	Cell content (usually character)
`theme`	Theme configuration
`cell_type`	Type of cell (static, computation, etc.)

Value

Formatted content

`format_console_content` *Format Content for Console Theme*

Description

Format Content for Console Theme

Usage

```
format_console_content(content, theme)
```

Arguments

`content`	Character content
`theme`	Theme configuration

Value

Formatted content

`format_content_for_output` *Format Content for Output Format*

Description

Format Content for Output Format

Usage

```
format_content_for_output(content, format, row, col, theme)
```

Arguments

`content`	Cell content
`format`	Output format
`row`	Row number
`col`	Column number
`theme`	Theme configuration

Value

Formatted content

`format_journal_content` *Format Content for Journal Themes*

Description

Format Content for Journal Themes

Usage

```
format_journal_content(content, theme)
```

Arguments

`content`	Character content
`theme`	Theme configuration

Value

Formatted content

`format_number` *Format Number with Appropriate Precision*

Description

Format Number with Appropriate Precision

Usage

```
format_number(x, digits = 3)
```

Arguments

`x`	Numeric value
`digits`	Maximum decimal places (default 3)

Value

Formatted string

`format_percentage` *Format Percentage*

Description

Format Percentage

Usage

```
format_percentage(proportion, digits = 1)
```

Arguments

`proportion`	Numeric proportion (0-1)
`digits`	Number of decimal places (default 1)

Value

Formatted percentage string

`format_pvalue` *Format P-value*

Description

Format P-value

Usage

```
format_pvalue(p)
```

Arguments

`p`	Numeric p-value

Value

Formatted string

`get_missing_summary` *Get Missing Data Summary*

Description

Get Missing Data Summary

Usage

```
get_missing_summary(data, vars = NULL)
```

Arguments

`data`	Data frame
`vars`	Character vector of variable names (default: all variables)

Value

Data frame with missing data information

`get_theme` *Get Theme Configuration*

Description

Get Theme Configuration

Usage

```
get_theme(theme_name = "console")
```

Arguments

`theme_name`	Character, theme name (console, nejm, lancet, jama, etc.)

Value

List with theme configuration

`is_empty` *Check if Object is Empty*

Description

Check if Object is Empty

Usage

```
is_empty(x)
```

Arguments

`x`	Object to check

Value

Logical indicating if object is empty

`list_available_themes` *Get Available Themes*

Description

Get Available Themes

Usage

```
list_available_themes()
```

Value

Character vector of available theme names

`parse_cell_key` *Parse Cell Key (Optimized)*

Description

Optimized key parsing using fast string manipulation instead of regex.

Usage

```
parse_cell_key(key)
```

Arguments

`key`	Character cell key (e.g., "r1_c2")

Value

List with integer row and col, or NA on failure

print.cell *Print Method for Cell Objects*

Description

Informative display of cell objects showing type and key content.

Usage

```
## S3 method for class 'cell'
print(x, ...)
```

Arguments

`x`	Cell object
`...`	Additional arguments

print.table1_blueprint *Enhanced Print Method for Blueprint*

Description

Provides informative console output for table1_blueprint objects including memory usage and population statistics.

Usage

```
## S3 method for class 'table1_blueprint'
print(x, ...)
```

Arguments

`x`	A table1_blueprint object
`...`	Additional arguments (unused)

Value

Invisibly returns the blueprint object

`print.table_dimensions` *Print Method for Optimized Dimensions*

Description

Clean display of dimension analysis results.

Usage

```
## S3 method for class 'table_dimensions'
print(x, ...)
```

Arguments

`x`	Table dimensions object
`...`	Additional arguments

`render_console` *Render Blueprint to Console*

Description

Render Blueprint to Console

Usage

```
render_console(blueprint, theme = NULL)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration (optional)

Value

Character vector for console output

`render_footnotes` *Render Footnotes*

Description

Render Footnotes

Usage

```
render_footnotes(blueprint, theme, format)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration
`format`	Output format

Value

Character vector with footnotes

`render_html` *Render Blueprint to HTML*

Description

Render Blueprint to HTML

Usage

```
render_html(blueprint, theme = NULL)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration (optional)

Value

Character vector with HTML code

`render_latex` *Render Blueprint to LaTeX*

Description

Render Blueprint to LaTeX

Usage

```
render_latex(blueprint, theme = NULL)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration (optional)

Value

Character vector with LaTeX code

`render_table_content` *Render Table Content (Optimized)*

Description

Optimized rendering logic that iterates over existing cells rather than all possible grid positions. This is much more efficient for the sparse tables generated by this package.

Usage

```
render_table_content(blueprint, theme, format)
```

Arguments

`blueprint`	Table1Blueprint object
`theme`	Theme configuration
`format`	Output format (console, latex, html)

Value

Character vector with table content

`safe_divide` *Safe Division*

Description

Safe Division

Usage

```
safe_divide(numerator, denominator, na_value = NA)
```

Arguments

`numerator`	Numeric numerator
`denominator`	Numeric denominator
`na_value`	Value to return if division by zero (default NA)

Value

Result of division or na_value

`summary.cell` *Summary Method for Cell Objects*

Description

Detailed summary of cell objects including metadata.

Usage

```
## S3 method for class 'cell'
summary(object, ...)
```

Arguments

`object`	Cell object
`...`	Additional arguments

`table1` *Backward Compatibility Wrapper*

Description

Provides backward compatibility with the original table1 function.

Usage

```
table1(form, data, ...)
```

Arguments

`form`	Formula
`data`	Data frame
`...`	All other arguments

Value

table1_blueprint object

`Table1Blueprint` *Create Memory-Efficient Table1 Blueprint Object*

Description

Creates an optimized blueprint object with sparse storage using R environments for hash-table like performance. Only populated cells consume memory, providing significant memory savings for typical sparse table structures.

Usage

```
Table1Blueprint(nrows, ncols)
```

Arguments

`nrows`	Integer specifying the number of rows in the final table. Must be a positive integer
`ncols`	Integer specifying the number of columns in the final table. Must be a positive integer

Details

The optimized blueprint uses environment-based sparse storage instead of pre-allocating all cells. Benefits include:

- Memory usage scales with actual content, not table dimensions
- O(1) hash-table lookup for cell access
- Automatic garbage collection of unused cells
- Support for very large sparse tables

Value

An object of class "table1_blueprint" with components:

- cells: Environment with hash-table storage for cells
- nrows: Number of rows
- ncols: Number of columns
- row_names: Character vector of row identifiers
- col_names: Character vector of column headers
- metadata: List containing structural information

See Also

Cell, validate_table1_blueprint

Examples

```
# Small table - minimal memory usage
bp_small <- Table1Blueprint(5, 3)
bp_small[1,1] <- Cell(type = "static", content = "Variable")

# Large sparse table - still efficient
bp_large <- Table1Blueprint(1000, 100)  # Only uses memory for metadata
```

---

table1_optimized *Create Publication-Ready Summary Tables (Optimized)*

Description

Optimized version of the table1 function with significant improvements in memory efficiency, error handling, and performance. Maintains full compatibility with the original interface while providing enhanced functionality.

Usage

```
table1_optimized(
  form,
  data,
  strata = NULL,
  block = NULL,
  missing = FALSE,
  pvalue = TRUE,
  size = FALSE,
  totals = FALSE,
  fname = "table1",
  layout = "console",
  numeric_summary = "mean_sd",
  footnotes = NULL,
  theme = "default",
  ...
)
```

Arguments

`form`	Formula specifying table structure (group ~ vars or ~ vars)
`data`	Data frame containing all variables
`strata`	Optional stratification variable name
`block`	Deprecated parameter (maintained for compatibility)
`missing`	Logical indicating whether to show missing value counts
`pvalue`	Logical indicating whether to include p-values
`size`	Logical indicating whether to show group sizes
`totals`	Logical indicating whether to include totals column
`fname`	Output filename (for export functions)
`layout`	Output format ("console", "latex", "html")
`numeric_summary`	Summary type for numeric variables
`footnotes`	Footnote specifications
`theme`	Journal theme ("default", "nejm", "lancet", "jama", "bmj")
`...`	Additional arguments for future extensibility

Details

This optimized version provides significant improvements:

- Memory efficiency: 60-80
- Performance: Vectorized operations and optimized algorithms
- Reliability: Comprehensive input validation and error handling
- Maintainability: Modular architecture with focused functions

Value

Optimized table1_blueprint object with sparse storage

Examples

```
## Not run:
# Basic usage
data(mtcars)
mtcars$transmission <- factor(ifelse(mtcars$am == 1, "Manual", "Auto"))

# Simple table
bp <- table1_optimized(transmission ~ mpg + hp, data = mtcars)
display_table(bp, mtcars)

# With theme and footnotes
bp <- table1_optimized(transmission ~ mpg + hp, data = mtcars,
                      theme = "nejm", pvalue = TRUE,
                      footnotes = list(
                        variables = list(mpg = "EPA fuel economy rating")
                      ))
display_table(bp, mtcars)

## End(Not run)
```

`validate_cell` *Validate Cell Object*

Description

Comprehensive validation of constructed cell objects ensuring structural integrity and type consistency.

Usage

```
validate_cell(x, strict = FALSE)
```

Arguments

`x`	Cell object to validate
`strict`	Logical indicating whether to perform expensive checks

Value

Validated cell object (invisibly) or stops with error

`validate_table1_blueprint` *Validate Table1 Blueprint Object*

Description

Comprehensive validation function that ensures blueprint objects maintain structural integrity and type safety.

Usage

```
validate_table1_blueprint(x, strict = FALSE)
```

Arguments

`x`	A table1_blueprint object to validate
`strict`	Logical indicating whether to perform expensive checks

Value

The validated object (invisibly) or stops with informative error

---

`validate_table1_inputs`

Validate Table1 Function Inputs

Description

Comprehensive validation function for table1() arguments that checks formula structure, data integrity, variable existence, and parameter consistency. Provides informative error messages to guide users.

Usage

```
validate_table1_inputs(
  formula,
  data,
  strata = NULL,
  theme = "default",
  layout = "console",
  numeric_summary = "mean_sd",
  footnotes = NULL,
  pvalue = TRUE,
  totals = FALSE,
  missing = FALSE,
  size = FALSE
)
```

Arguments

`formula`	Formula object specifying table structure
`data`	Data frame containing analysis variables
`strata`	Character string naming stratification variable (optional)
`theme`	Character string specifying theme name
`layout`	Character string specifying output layout
`numeric_summary`	Summary specification for numeric variables
`footnotes`	List containing footnote specifications
`pvalue`	Logical indicating whether to include p-values
`totals`	Logical indicating whether to include totals
`missing`	Logical indicating whether to show missing counts
`size`	Logical indicating whether to show group sizes

Details

The validation process includes:

- Formula structure and variable existence
- Data frame integrity and type checking
- Parameter consistency and logical constraints
- Performance warnings for large datasets
- Data quality assessments with recommendations

Value

Invisibly returns TRUE if all validations pass

Examples

```
## Not run:
# Valid inputs
validate_table1_inputs(mpg ~ cyl, mtcars, pvalue = TRUE)

# Will error - missing variable
validate_table1_inputs(mpg ~ missing_var, mtcars)

## End(Not run)
```

`validate_theme` *Validate Theme Configuration*

Description

Validate Theme Configuration

Usage

```
validate_theme(theme)
```

Arguments

`theme`	Theme configuration list

Value

TRUE if valid, stops with error if invalid

`[.table1_blueprint` *Optimized Cell Access for Blueprint*

Description

Provides efficient matrix-like indexing for table1_blueprint objects using environment-based hash table lookup.

Usage

```
## S3 method for class 'table1_blueprint'
x[i, j, drop = FALSE]
```

Arguments

`x`	A table1_blueprint object
`i`	Row index (1-based)
`j`	Column index (1-based)
`drop`	Logical (ignored for compatibility)

Details

The optimized implementation uses O(1) hash table lookup through R environments. Bounds checking is performed to ensure safe access.

Value

The cell object at position [i, j] or NULL if empty

Examples

```
bp <- Table1Blueprint(5, 3)
bp[1, 1] <- Cell(type = "static", content = "Variable")
cell <- bp[1, 1]  # O(1) lookup
```

`[<-.table1_blueprint` *Optimized Cell Assignment for Blueprint*

Description

Provides efficient assignment of cells to blueprint positions using hash table storage with automatic memory management.

Usage

```
## S3 replacement method for class 'table1_blueprint'
x[i, j] <- value
```

Arguments

`x`	A table1_blueprint object
`i`	Row index (1-based)
`j`	Column index (1-based)
`value`	A Cell object or NULL to remove

Details

Assignment automatically manages memory by:

- Storing only non-NULL cells
- Removing cells when assigned NULL
- Updating cell count metadata
- Validating cell objects before storage

Value

Modified table1_blueprint object

Examples

```
bp <- Table1Blueprint(5, 3)

# Assign cell
bp[1, 1] <- Cell(type = "static", content = "Variable")

# Remove cell
bp[1, 1] <- NULL
```

Index