Creating LaTeX Tables and Cropped PDFs from Dataframes

Ronald (Ryy) G. Thomas

Introduction

The `zztab2fig` package provides a streamlined workflow to transform R dataframes into publication-ready LaTeX tables and generate cropped PDF outputs. With built-in options for styling, directory management, and verbose logging, the package is ideal for researchers, analysts, and data scientists who need professional table outputs for reports, presentations, or publications.

This vignette will cover:

1. Installing and loading the package.
2. Key functions and their usage.
3. A complete example of generating LaTeX tables and PDFs.
4. Customization options.
5. Troubleshooting and tips.

Installation

The package can be installed from CRAN or directly from GitHub if hosted there.

```
# Install from CRAN
install.packages("zztab2fig")

# Or install from GitHub
# devtools::install_github("yourusername/zztab2fig")
```

Load the package:

```
library(zztab2fig)
# Load dplyr for data manipulation examples
library(dplyr)
```

Key Functions

`t2f()`

The primary function for converting a dataframe to a LaTeX table and generating a cropped PDF.

Arguments

- `df`: A dataframe to be converted.
- `filename`: Base name of the output files (without extensions). Defaults to the dataframe name.
- `sub_dir`: Subdirectory to store the output files. Default: `"output"`.
- `scolor`: A LaTeX color for alternating row shading (e.g., `"blue!10"`). Default: `"blue!10"`.
- `verbose`: Logical. If `TRUE`, prints progress messages. Default: `FALSE`.

- `extra_packages`: A list of LaTeX package specifications. Can include helper functions or raw LaTeX strings. Default: `NULL`.
- `document_class`: LaTeX document class to use. Default: `"article"`.

Returns

- Invisibly returns the path to the cropped PDF.

LaTeX Package Helper Functions

The package provides convenient R functions to generate LaTeX package specifications:

`geometry()`

Configures page layout and margins:

```
geometry(margin = "5mm", paper = "a4paper", landscape = TRUE)
```

`babel()`

Adds language support:

```
babel("spanish")  # for Spanish language support
```

`fontspec()`

Configures custom fonts (requires XeLaTeX or LuaLaTeX):

```
fontspec(main_font = "Times New Roman", sans_font = "Arial")
```

Helper Functions

See the full vignette for detailed descriptions of helper functions like `sanitize_column_names()`, `sanitize_table_cells()`, and `create_latex_table()`.

Example Workflow

Step 1: Prepare Your Data

Load a dataframe, e.g., the `mtcars` dataset:

```
data(mtcars)
```

Step 2: Generate the LaTeX Table and PDF

```
t2f(
  df = mtcars,
  filename = "mtcars_table",
  sub_dir = "tables",
  scolor = "blue!10",
  verbose = TRUE
)
```

Step 3: Advanced Usage with Custom LaTeX Packages

```
# Create a table with custom margins and language support
t2f(
  df = mtcars,
  filename = "mtcars_custom",
  sub_dir = "tables",
  extra_packages = list(
    geometry(margin = "5mm", paper = "a4paper"),
    babel("spanish")
  ),
  verbose = TRUE
)

# Wide table with landscape orientation
t2f(
  df = mtcars,
  filename = "mtcars_wide",
  extra_packages = list(
    geometry(margin = "3mm", landscape = TRUE)
  )
)
```

Step 4: Inspect the Outputs

Navigate to the `tables` subdirectory to find: - `mtcars_table.tex`: The LaTeX source file. - `mtcars_table.pdf`: The compiled PDF. - `mtcars_table_cropped.pdf`: The cropped PDF.

Feature Demonstrations

Basic Table Generation

The simplest usage creates a standard LaTeX table:

```
library(zztab2fig)

# Basic table with default settings
data(iris)
small_iris <- head(iris, 8)

result <- t2f(small_iris,
              filename = "basic_iris",
              sub_dir = "basic_output",
              verbose = TRUE)
cat("Generated files in:", dirname(result))
```

```
## Generated files in: basic_output
```

This creates: - `basic_output/basic_iris.tex`: LaTeX source file - `basic_output/basic_iris.pdf`: Full PDF - `basic_output/basic_iris_cropped.pdf`: Cropped PDF for inclusion

Here's the table content that was generated:

Table 1: Basic Iris Dataset Table (as displayed in R)

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa

And here's the actual LaTeX-generated PDF table produced by `zztab2fig`:

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa

Figure 1: LaTeX Table Generated by zztab2fig

Custom Colors and Styling

Control table appearance with different color schemes. Here's a comparison:

```
# Create a sample dataset
color_demo_data <- head(mtcars[, c("mpg", "cyl", "hp", "wt")], 6)

# Generate tables with different colors
t2f(color_demo_data,
    filename = "blue_table",
    sub_dir = "color_demo",
    scolor = "blue!12")

t2f(color_demo_data,
    filename = "green_table",
    sub_dir = "color_demo",
    scolor = "green!15")
```

Original data:

Table 2: Sample Data for Color Demonstration

	mpg	cyl	hp	wt
Mazda RX4	21.0	6	110	2.620
Mazda RX4 Wag	21.0	6	110	2.875
Datsun 710	22.8	4	93	2.320
Hornet 4 Drive	21.4	6	110	3.215
Hornet Sportabout	18.7	8	175	3.440
Valiant	18.1	6	105	3.460

Blue-styled LaTeX table (scolor = "blue!12"):

	mpg	**cyl**	**hp**	**wt**
Mazda RX4	21.0	6	110	2.620
Mazda RX4 Wag	21.0	6	110	2.875
Datsun 710	22.8	4	93	2.320
Hornet 4 Drive	21.4	6	110	3.215
Hornet Sportabout	18.7	8	175	3.440
Valiant	18.1	6	105	3.460

Figure 2: Blue-styled LaTeX Table

Green-styled LaTeX table (scolor = "green!15"):

Available color options include: `blue!10`, `green!20`, `red!15`, `gray!25`, `purple!12`, `orange!18`, `yellow!30`, etc. The number after `!` controls intensity (lower = lighter).

Page Layout Customization

Minimal Margins for Wide Tables

```
# Perfect for wide datasets
wide_data <- mtcars[1:8, 1:8]  # First 8 rows and columns

t2f(wide_data,
    filename = "wide_table",
    extra_packages = list(
      geometry(margin = "3mm")
    ))
```

	mpg	cyl	hp	wt
Mazda RX4	21.0	6	110	2.620
Mazda RX4 Wag	21.0	6	110	2.875
Datsun 710	22.8	4	93	2.320
Hornet 4 Drive	21.4	6	110	3.215
Hornet Sportabout	18.7	8	175	3.440
Valiant	18.1	6	105	3.460

Figure 3: Green-styled LaTeX Table

Landscape Orientation

```
# Landscape mode for very wide tables
t2f(mtcars,
    filename = "landscape_mtcars",
    extra_packages = list(
      geometry(margin = "5mm", landscape = TRUE)
    ))
```

Custom Paper Sizes

```
# A4 paper with specific margins
t2f(small_iris,
    filename = "a4_table",
    extra_packages = list(
      geometry(paper = "a4paper", margin = "10mm")
    ))

# Letter paper
t2f(small_iris,
    filename = "letter_table",
    extra_packages = list(
      geometry(paper = "letterpaper", margin = "0.75in")
    ))
```

Advanced Use Cases

Multilingual Documents

Create tables with proper language support:

```r
# Spanish language support
spanish_data <- data.frame(
  "Producto" = c("Café", "Té", "Azúcar"),
  "Precio" = c("$2.50", "$1.80", "$0.75"),
  "Disponible" = c("Sí", "Sí", "No"),
  stringsAsFactors = FALSE
)

t2f(spanish_data,
    filename = "tabla_espanol",
    extra_packages = list(
      babel("spanish")
    ))

# French with custom margins
french_data <- data.frame(
  "Produit" = c("Café", "Thé", "Sucre"),
  "Prix" = c("2,50€", "1,80€", "0,75€"),
  stringsAsFactors = FALSE
)

t2f(french_data,
    filename = "tableau_francais",
    extra_packages = list(
      babel("french"),
      geometry(margin = "8mm")
    ))
```

Document Classes

Different document classes for specific purposes:

```r
# Minimal document class (less overhead)
t2f(small_iris,
    filename = "minimal_table",
    document_class = "minimal",
    extra_packages = list(
      geometry(margin = "5mm")
    ))

# Article class with custom formatting
t2f(small_iris,
    filename = "article_table",
    document_class = "article")
```

Complex Layouts

Combine multiple LaTeX packages for sophisticated layouts:

```r
# Professional report-style table
report_data <- mtcars[1:10, c("mpg", "cyl", "disp", "hp", "wt")]

t2f(report_data,
    filename = "professional_table",
```

```
    scolor = "blue!8",
    extra_packages = list(
      geometry(margin = "15mm", paper = "a4paper"),
      "\\usepackage{microtype}",  # Better typography
      "\\usepackage{array}"       # Enhanced table formatting
    ))

# Conference poster table (large, landscape)
t2f(mtcars,
    filename = "poster_table",
    scolor = "blue!12",
    extra_packages = list(
      geometry(
        paper = "a3paper",
        margin = "10mm",
        landscape = TRUE
      )
    ))
```

Data Processing Workflows

Working with dplyr

```
library(dplyr)

# Summary statistics table
summary_stats <- mtcars %>%
  group_by(cyl) %>%
  summarise(
    Count = n(),
    Mean_MPG = round(mean(mpg), 2),
    Mean_HP = round(mean(hp), 1),
    .groups = "drop"
  )

t2f(summary_stats,
    filename = "cylinder_summary",
    scolor = "green!12")

# Top performers table
top_mpg <- mtcars %>%
  arrange(desc(mpg)) %>%
  slice_head(n = 8) %>%
  select(mpg, cyl, hp, wt)

t2f(top_mpg,
    filename = "fuel_efficient_cars",
    scolor = "blue!15")
```

Handling Special Characters

The package automatically sanitizes LaTeX special characters:

```r
# Data with special characters
special_data <- data.frame(
  "Category" = c("R&D", "Sales", "Marketing"),
  "Budget%" = c("25%", "40%", "35%"),
  "Cost$" = c("$50K", "$80K", "$70K"),
  "Notes#" = c("Priority #1", "High #2", "Medium #3"),
  stringsAsFactors = FALSE
)

t2f(special_data,
    filename = "special_chars_demo",
    scolor = "orange!15")
```

Batch Processing

Generate multiple tables efficiently:

```r
# Process multiple datasets
datasets <- list(
  cars = mtcars[1:10, 1:5],
  flowers = iris[1:10, ],
  trees = trees[1:10, ]
)

# Generate all tables with consistent styling
for (name in names(datasets)) {
  t2f(datasets[[name]],
      filename = paste0("batch_", name),
      scolor = "blue!10",
      sub_dir = "batch_output")
}

# Or using lapply
lapply(names(datasets), function(name) {
  t2f(datasets[[name]],
      filename = name,
      sub_dir = paste0("dataset_", name))
})
```

Integration Strategies

With R Markdown

Include generated PDFs in R Markdown documents. Here's a working example:

```r
library(dplyr)
library(zztab2fig)

# Generate summary statistics table
summary_data <- mtcars %>%
  group_by(cyl) %>%
  summarise(
    Count = n(),
    Mean_MPG = round(mean(mpg), 1),
    Mean_HP = round(mean(hp), 0),
```

```
    .groups = "drop"
  )

# Create the table
result_path <- t2f(summary_data, "vignette_demo", sub_dir = "tables")
cat("Table generated at:", result_path)
```

```
## Table generated at: tables/vignette_demo_cropped.pdf
```

The table is now saved as a PDF that can be included in documents or presentations.

Here's what the data looks like in R:

Table 3: Summary Data (as displayed in R)

cyl	Count	Mean_MPG	Mean_HP
4	11	26.7	83
6	7	19.7	122
8	14	15.1	209

And here's the professional LaTeX table generated by `t2f()`:

cyl	Count	Mean_MPG	Mean_HP
4	11	26.7	83
6	7	19.7	122
8	14	15.1	209

Figure 4: Professional LaTeX Table from t2f()

With LaTeX Documents

Use generated `.tex` files in larger documents:

```
\documentclass{article}
\usepackage{graphicx}

\begin{document}
\section{Data Analysis}

The following table shows our results:

\input{tables/results.tex}

\end{document}
```

With Shiny Applications

```
# In server.R
output$download_table <- downloadHandler(
  filename = "analysis_table.pdf",
  content = function(file) {
    # Generate table
    result <- t2f(processed_data(),
                  filename = "shiny_table",
                  sub_dir = tempdir())
    file.copy(result, file)
  }
)
```

Troubleshooting & Best Practices

Common Installation Issues

Error: `pdflatex` or `pdfcrop` not found

Solution by Platform: - **Ubuntu/Debian:** `sudo apt-get install texlive texlive-extra-utils` - **macOS:** Install MacTeX from https://www.tug.org/mactex/ - **Windows:** Install MiKTeX from https://miktex.org/ - **Conda:** `conda install -c conda-forge texlive-core`

Verify Installation:

```
pdflatex --version
pdfcrop --version
```

LaTeX Compilation Errors

The package now provides detailed error messages when LaTeX compilation fails:

```
# If this fails, you'll get specific error information
problematic_data <- data.frame(
  "Bad\\Column" = 1:3,  # Unescaped backslash
  stringsAsFactors = FALSE
)

# The error message will show LaTeX log details
t2f(problematic_data, "test_error")
```

Debugging Steps: 1. Check the generated `.tex` file for syntax errors 2. Look at the `.log` file in the output directory
3. Test the `.tex` file manually with `pdflatex` 4. Verify your data doesn't contain problematic characters

Performance Tips

For Large Tables

```
# Memory-efficient processing for large datasets
large_data <- data.frame(
  matrix(rnorm(10000), ncol = 10)
)

# Use minimal document class to reduce overhead
```

```r
t2f(large_data,
    filename = "large_table",
    document_class = "minimal",
    extra_packages = list(
      geometry(margin = "2mm", landscape = TRUE)
    ))
```

Batch Processing Optimization

```r
# Process multiple tables efficiently
# datasets <- list(...)  # Your actual datasets here

# Set up common directory first
output_dir <- "batch_tables"
dir.create(output_dir, showWarnings = FALSE)

# Use consistent settings to avoid repeated LaTeX compilation setup
common_settings <- list(
  sub_dir = output_dir,
  scolor = "blue!10",
  verbose = FALSE  # Reduce console output for batch processing
)

# Process with do.call to avoid repetitive code
lapply(names(datasets), function(name) {
  do.call(t2f, c(list(df = datasets[[name]], filename = name), common_settings))
})
```

Best Practices

Data Preparation

```r
library(dplyr)
library(stringr)

# Clean your data before table generation
prepare_table_data <- function(df) {
  df %>%
    # Round numeric columns appropriately
    mutate(across(where(is.numeric), ~ round(.x, 2))) %>%
    # Handle NA values
    mutate(across(everything(), ~ ifelse(is.na(.x), "-", as.character(.x)))) %>%
    # Ensure reasonable column names
    rename_with(~ gsub("\\.", " ", .x)) %>%
    # Limit precision for display
    mutate(across(where(is.character), ~ str_trunc(.x, 20)))
}

clean_data <- prepare_table_data(messy_data)
t2f(clean_data, filename = "clean_table")
```

Consistent Styling

Create reusable style configurations:

```
# Define style presets
table_styles <- list(
  corporate = list(
    scolor = "blue!8",
    extra_packages = list(
      geometry(margin = "15mm", paper = "a4paper")
    )
  ),

  presentation = list(
    scolor = "blue!15",
    extra_packages = list(
      geometry(margin = "5mm", landscape = TRUE)
    )
  ),

  minimal = list(
    scolor = "gray!10",
    document_class = "minimal",
    extra_packages = list(geometry(margin = "3mm"))
  )
)

# Apply consistent styling
apply_style <- function(data, filename, style_name) {
  style <- table_styles[[style_name]]
  do.call(t2f, c(list(df = data, filename = filename), style))
}

# Usage
apply_style(mtcars, "corporate_mtcars", "corporate")
apply_style(iris, "presentation_iris", "presentation")
```

File Organization

```
# Organize outputs by project/analysis
project_name <- "quarterly_analysis"

# Create structured output directories
output_structure <- c(
  file.path(project_name, "tables"),
  file.path(project_name, "figures"),
  file.path(project_name, "temp")
)

sapply(output_structure, dir.create, recursive = TRUE, showWarnings = FALSE)

# Generate tables with organized structure
t2f(quarterly_data,
    filename = "q4_summary",
```

```
    sub_dir = file.path(project_name, "tables"))
```

Advanced LaTeX Customization

Custom LaTeX Preambles

```
# For users comfortable with LaTeX
custom_preamble <- list(
  "\\usepackage{booktabs}",
  "\\usepackage{longtable}",
  "\\usepackage{microtype}",
  "\\renewcommand{\\arraystretch}{1.2}",  # Increase row spacing
  "\\usepackage[table]{xcolor}"
)

# t2f(your_data,
#     filename = "custom_latex",
#     extra_packages = custom_preamble)
```

Integration with External LaTeX Templates

```
# Generate just the table content
# table_only_data <- your_data  # Your actual data here

# Create table with minimal wrapper for inclusion
# t2f(table_only_data,
#     filename = "table_content",
#     document_class = "minimal",
#     extra_packages = list("\\usepackage{booktabs}"))

# Then include the generated .tex content in your main document
```

Real-World Examples

Academic Papers

```
# Create publication-ready tables
results_table <- data.frame(
  Model = c("Linear", "Quadratic", "Cubic"),
  R_squared = c(0.85, 0.92, 0.94),
  AIC = c(145.2, 138.7, 141.3),
  p_value = c("< 0.001", "< 0.001", "0.002"),
  stringsAsFactors = FALSE
)

# Academic journal formatting
t2f(results_table,
    filename = "model_comparison",
    scolor = "gray!8",
    extra_packages = list(
      geometry(margin = "20mm", paper = "letterpaper")
    ))
```

Business Reports

```
# Quarterly sales summary
sales_summary <- data.frame(
  Quarter = c("Q1", "Q2", "Q3", "Q4"),
  Revenue = c("$125K", "$143K", "$158K", "$171K"),
  Growth = c("-", "+14.4%", "+10.5%", "+8.2%"),
  Target = c("$120K", "$140K", "$155K", "$165K"),
  Status = c(" ", " ", " ", " "),
  stringsAsFactors = FALSE
)

t2f(sales_summary,
    filename = "quarterly_sales",
    scolor = "blue!10",
    extra_packages = list(
      geometry(margin = "12mm", paper = "a4paper")
    ))
```

Conference Presentations

```
# Large, readable table for presentations
key_findings <- mtcars[1:6, c("mpg", "cyl", "hp", "wt")]

t2f(key_findings,
    filename = "presentation_findings",
    scolor = "blue!15",
    extra_packages = list(
      geometry(
        margin = "8mm",
        paper = "a4paper",
        landscape = TRUE
      )
    ))
```

References

- **kableExtra**: A package for creating and styling LaTeX/HTML tables in R. CRAN link
- **pdflatex and pdfcrop**: Tools from TeX distributions like TeX Live and MiKTeX.
- **zztab2fig**: GitHub link (if applicable).

Conclusion

The zztab2fig package simplifies the process of creating professional-quality tables for publications and presentations. By combining R's data processing capabilities with LaTeX's typesetting power, the package bridges the gap between analysis and presentation, making it an essential tool for any data analyst or researcher.