Paper OS21

A Sample Paper for a PHUSE Author

Mike Stackhouse, Atorus Research, Chapel Hill, NC, USA
Second Author, Company, City, Country

ABSTRACT

Over the past few years, the industry has made significant strides by releasing open-source R packages designed specifically for generating regulatory-compliant tables. Despite the availability of various options, organizations venturing into this territory often discover that their Tables, Listings, and Figures (TLF) requirements are not fully satisfied. To bridge this gap, we introduced clinify—a new R package that enhances and encapsulates functionalities from two robust R packages: flextable and officer.

This presentation aims to shed light on several key points: our rationale behind developing a new package when numerous solutions already exist; the additional requirements and functionalities clinify meets; and our method of achieving these goals. We adopted an open-source philosophy for clinify, focusing on expanding upon an existing foundation rather than starting from scratch. Join us as we delve into why clinify represents a crucial evolution in meeting TLF needs with efficiency and flexibility.

INTRODUCTION

The primary motivation behind **{clinify}** is to take the things that are great about the R packages {flextable} and {officer}, take the standard and complex pieces of formatting clinical tables for regulatory use, and simplify the tedious pieces. {flextable} and {officer} offer a huge range of capability for creating tables in R and rendering them to various formats. {flextable} makes formatting the table itself straightforward, while {officer} gives you lower level access to create documents like docx files and insert separate components as needed.

When working with clinical tables, the devil is in the details. Every organization tends to have its own bits of nuance, and the flexibility of each organization to deviate from those standards varies. In the R world, there have still been a number of features that have either made generating clinical tables with certain features very tedious, or in some cases not possible with the current tooling. With **{clinify}** we attempt to close that gap and give some quality-of-life features to programmers making these tables.

BASICS

Let's start at the beginning

```
ct <- clintable(mtcars)
print(ct)
```

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4

In **{clinify}**, a clintable itself is at it's root a flextable object with some extra metadata attached to it. A core part of the design philosophy of **{clinify}** is to build off of {flextable} at its core, extending functionality so that {flextable} functions are still operable on a clintable object.

The table printed above is the foundation of a clintable object. What we see here is the print method of a clintable being used. Compared to flextable, the primary thing that has is the application of default styling. Organizations generally have specific style preferences for their outputs, such as font and font size, standard conventions for borders, page size and margins, etc. These are configuration in **{clinify}** using some standard options that will be explained in another vignette. The print() method respects these settings to allow you to interactively explore your table being formatted.

TITLES AND FOOTNOTES

Let's expand some features of the table.

```
ct <- clintable(mtcars) |>
  clin_add_titles(
    list(
      c("Left", "Center", "Right"),
      c("Just the middle")
    )
  ) |>
  clin_add_footnotes(
      list(
        c(
          "Here's a footnote.",
          format(Sys.time(), "%H:%M %A, %B %d, %Y")
        )
      )
    )


print(ct)
```

Left | Center | Right

Just the middle

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4

Here's a footnote. 15:07 Tuesday, March 04, 2025

Here we've added some titles and footnotes to the document. The functions clin_add_titles() and clin_add_footnotes() allow you to insert titles and footnotes into the clintable metadata. When the clintable is written to a document or printed, the titles are respected. When you're printing interactively, the HTML that's rendered allows you to see the titles and footnotes above and below the table as if you're viewing an individual page. When writing to a docx file, the titles are placed in the header and the footnotes are placed into the footer.

In this example, by providing a list of character vectors, each element of the list is added as a new line. There are some broad assumptions being made:

- In titles, a single element will align center. In footnotes, the element will align right.
- If two elements are provided, they align left and right.
- If three elements are provided, they align left, right, and center.

Ultimately, the attached tables are converted to flextables. As such, you can create your own flextable and attach it to the header or footer using the ft option in clin_add_titles(). We also have the helper function new_title_footnote() that allows you to supply a list and generate the flextable so you can apply extra formatting as desired.

PAGINATION AND ALTERNATING PAGES

Let's look at a couple more functions.

```
dat <- mtcars
dat['page'] <- c(
  rep(1, 10),
  rep(2, 10),
  rep(3, 10),
  c(4, 4)
)
dat2 <- rbind(dat, dat)
dat2['groups1'] <- c(
  rep('a', 32),
  rep('b', 32)
)
dat2['groups2'] <- c(
  rep('1', 16),
  rep('2', 16),
  rep('1', 16),
  rep('2', 16)
)

# Create a basic table
ct <- clintable(dat2) |>
  clin_page_by('page') |>
  clin_group_by(c('groups1', 'groups2')) |>
  clin_alt_pages(
    key_cols = c('mpg', 'cyl', 'hp'),
    col_groups = list(
      c('disp', 'drat', 'wt'),
      c('qsec', 'vs', 'am'),
      c('gear', 'carb')
    )
  ) |>
  clin_col_widths(mpg = .2, cyl=.2, disp=.15, vs=.15) |>
  clin_add_titles(
    list(
      c("Left", "Center", "Right"),
      c("Just the middle")
    )
  ) |>
  clin_add_footnotes(
      list(
        c(
          "Here's a footnote.",
          format(Sys.time(), "%H:%M %A, %B %d, %Y")
        )
```

```
      )
    )

print(ct)
```

Left	Center	Right
	Just the middle	

a
1

	mpg	cyl	hp	disp	drat	wt
	21.0	6	110	160.0	3.90	2.620
	21.0	6	110	160.0	3.90	2.875
	22.8	4	93	108.0	3.85	2.320
	21.4	6	110	258.0	3.08	3.215
	18.7	8	175	360.0	3.15	3.440
	18.1	6	105	225.0	2.76	3.460
	14.3	8	245	360.0	3.21	3.570
	24.4	4	62	146.7	3.69	3.190
	22.8	4	95	140.8	3.92	3.150
	19.2	6	123	167.6	3.92	3.440

Here's a footnote. 15:07 Tuesday, March 04, 2025

1 2 3

Left	Center	Right
	Just the middle	

a
1

	mpg	cyl	hp	qsec	vs	am
	21.0	6	110	16.46	0	1
	21.0	6	110	17.02	0	1
	22.8	4	93	18.61	1	1
	21.4	6	110	19.44	1	0
	18.7	8	175	17.02	0	0
	18.1	6	105	20.22	1	0
	14.3	8	245	15.84	0	0
	24.4	4	62	20.00	1	0
	22.8	4	95	22.90	1	0
	19.2	6	123	18.30	1	0

Here's a footnote. 15:07 Tuesday, March 04, 2025

1 2 3

Left Center Right

Just the middle

a
1

mpg	cyl	hp	qsec	vs	am
21.0	6	110	16.46	0	1
21.0	6	110	17.02	0	1
22.8	4	93	18.61	1	1
21.4	6	110	19.44	1	0
18.7	8	175	17.02	0	0
18.1	6	105	20.22	1	0
14.3	8	245	15.84	0	0
24.4	4	62	20.00	1	0
22.8	4	95	22.90	1	0
19.2	6	123	18.30	1	0

Here's a footnote. 15:07 Tuesday, March 04, 2025

1 2 3

A number of new things have happened here. Let's go through function by function.

First, we've used the function clin_page_by() manually specify how page breaks should be handled. This is a data driven function, so here we've specified to use the page variable from the dat2 dataframe. Each time this variable changes, a page break will be inserted. Note that this isn't used for sorting, just inserting a change break.

Next, we've used the function clin_group_by(). This allows us to put by lines above the column headers using data from the input data frame. Similar to clin_page_by() each time this value changes, a new page will start. Just like the table data, these lines above the column headers will always reflect the data from within the variable. You can use as many group variable as you need here.

Next we've used clin_alt_pages(). This has been one of the most requested features we've seen since we originally developed the package {pharmaRTF}. This feature is designed to handle cases where the number of variables you need to present overflow the width of the page that you have available. The function works with rotating pages, so the same data rows are presented for each overflowing page, necessary, while the columns being presented change. After the first input of the clintable object, you have two parameters:

- key_cols: Columns that should be fixed to each page being presented
- col_groups: The groups of columns that should be presented on each of the alternating pages

If we look at that function specifically:

```
[...] |>
clin_alt_pages(
  key_cols = c('mpg', 'cyl', 'hp'),
  col_groups = list(
    c('disp', 'drat', 'wt'),
    c('qsec', 'vs', 'am'),
    c('gear', 'carb')
  )
)
```

In total there will be 3 alternating pages. The columns presented on each page will be:

- mpg, cyl, hp, disp, drat, wt
- mpg, cyl, hp, qsec, vs, am
- mpg, cyl, hp, gear, and carb

To make things easier for the developer, while developing interactively the print method has been updated to print 3 pages as styled HTML into the viewer pane, where you can select the page of choice from a page selector below the table. When printing to a word document, pages are inserted in the proper order. The logic works as follows:

- Identify chunk of rows for a given page. This can be based on the page_by method to manually insert page breaks or you can select maximum rows to print to a single page
- Identify the separate columns necessary for the individual pages
- Write out the chunk of rows for each set of col_groups in order before jumping to the next set of rows.

COLUMN HEADERS AND WIDTHS

From the previous example, another **{clinify}** function we used was clin_col_width(). The goal of this function is simply to make setting your column widths for a table straightforward. By default, in {flextable} your column widths are based on a unit such as inches or centimeters. In clin_col_width() we allow you to use the proportion of the page that you'd like that column to fill. From the syntax above:

```
[...] |>
  clin_col_widths(mpg = .2, cyl=.2, disp=.15, vs=.15) |>
[...]
```

In this case, we're saying that:

- mpg will fill 20% of the page
- cyl will fill 20% of the page
- disp will fill 15% of the page
- vs will fill 15% of the page

The rest of the columns will be spaced evenly based on the remaining space. The space that's filled is based on default configurations for page width, which are configurable within your session.

Furthermore, clin_col_width() adapts to alternating pages. The ratios given to key columns apply for each alternating page, and the proportions applied to the additional col_group variables adapt any remaining space to ensure a page fits the total page width.

One last tedious part of structuring any table is getting the table headers formatted correctly. There are a couple specific features, such as spanning headers, which can also be tricky to get right, especially in a semi-automated way. For this reason, we've added the function clin_column_headers() to make this process a bit easier. Let's use iris as an example of a table to which we want to apply some spanning headers.

```
clintable(iris) |>
  clin_column_headers(
    Sepal.Length = c("Flowers", "Sepal", "Length"),
    Sepal.Width = c("Flowers", "Sepal", "Width"),
    Petal.Length = c("Petal", "Length"),
    Petal.Width = c("Petal", "Width")
  )
```

Flowers				
Sepal		Petal		
Length	Width	Length	Width	
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
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa

The first parameter of clin_column_headers() will be the clintable object for which you want headers to apply. From there, use the column name to which you're applying a header. The way this function works is to use a character element for row of headers you want to apply. So for example, if you need three rows of headers, you can use 3 elements for a single column. The elements go to their respective rows.

When using spanning headers, you're also typically using cell merging so that a single string of text spans over multiple columns. To accomplish this, repeat the text that you want to merge and ensure that those elements are placed in the same row. Consider the example above:

```
[...]
      Sepal.Length = c("Flowers", "Sepal", "Length"),
      Sepal.Width = c("Flowers", "Sepal", "Width"),
[...]
```

For the Sepal variable, we have two spanning headers. One for "flowers", and one for "Sepal". These cells will be merged, and the bottom row contains "Length" and "Width" separately.

Another common way you may want to apply headers is by using your variable labels. By default, clintable will respect this. Furthermore, the same spanning can be achieved as well. Let's consider another example:

```
iris2 <- iris
attr(iris2$Sepal.Length, 'label') <- "Flower||Sepal||Length"
attr(iris2$Sepal.Width, 'label') <- "Flower||Sepal||Width"
attr(iris2$Petal.Length, 'label') <- "Flower||Petal||Length"
attr(iris2$Petal.Width, 'label') <- "Flower||Petal||Width"

clintable(iris2) |>
  align(align='center', part='header') |>
  align(align='center', part='body')
```

Flower				
Sepal		Petal		
Length	Width	Length	Width	
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
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa

The underlying logic of this example is exactly the same as using clin_column_headers(), and in fact clin_column_headers() is actually called by default. The difference is that here, to separate levels we use the delimiter ||. Note in this example how the spanning variable have also changed so that Flower stretches over all four Petal and Sepal columns.

Note that after headers are applied, additional styling can be done. In this case, we use the flextable::align() function to change the alignment on both the table body and column headers to center. Note that default styles are applied when the table is written out or printed, so these might potentially override some settings depending on how those functions are applied.

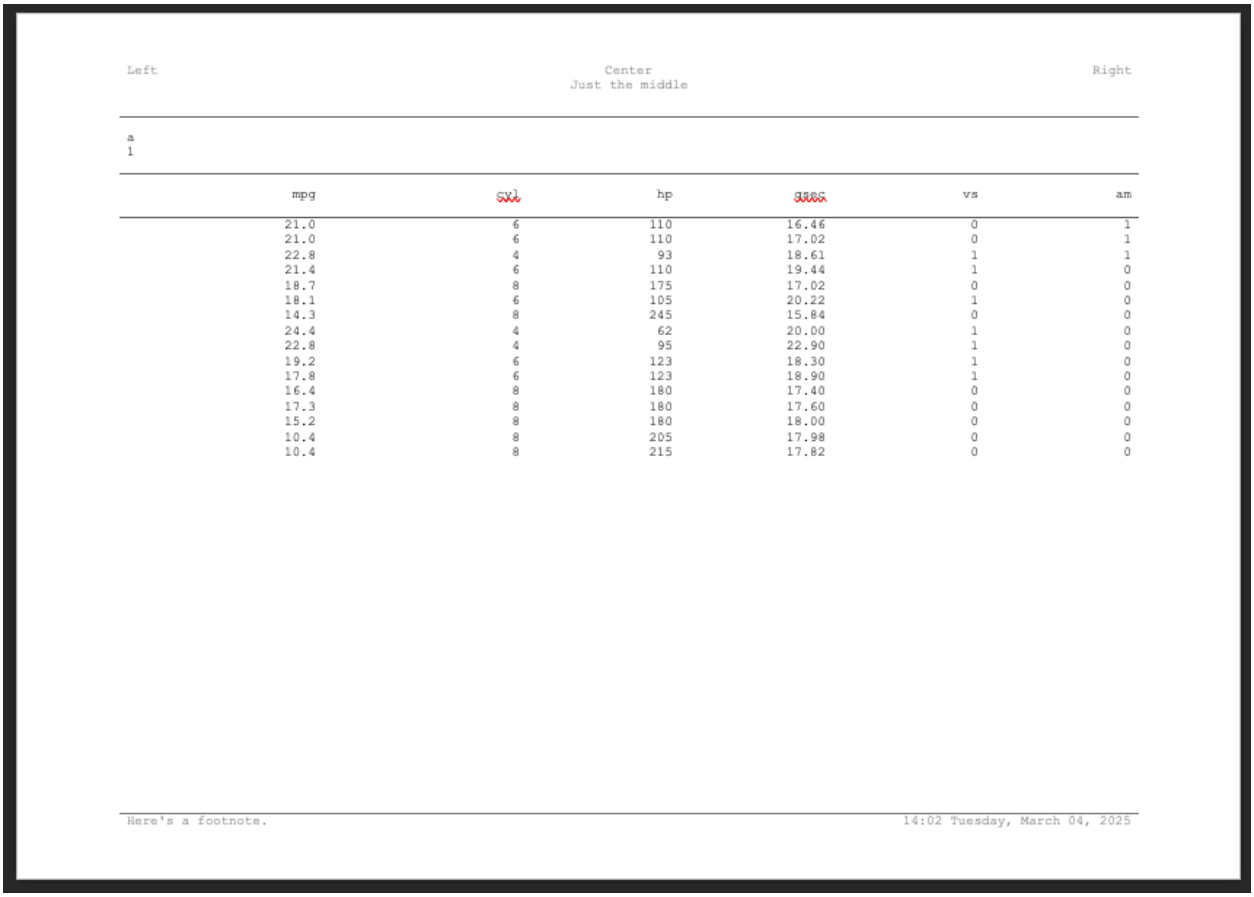
WRITING TO DOCX

While printing the table during development helps you with the development process, ultimately **{clinify}** lets you write the document to a docx file. This was a primary reason why we wanted to build on top of {flextable}; with {flextable} and {officer} we're able to have a great amount of control over how things are written specifically written into the word document. We try to make this process rather seamless and comparable to the print() method. To write the document out to docx, use the write_clintable() function. Let's revisit our table from before.

```
# Create a basic table
ct <- clintable(dat2) |>
  clin_page_by(max_rows = 36) |>
  clin_group_by(c('groups1', 'groups2')) |>
  clin_alt_pages(
    key_cols = c('mpg', 'cyl', 'hp'),
    col_groups = list(
      c('disp', 'drat', 'wt'),
      c('qsec', 'vs', 'am'),
      c('gear', 'carb')
    )
  ) |>
  clin_col_widths(mpg = .2, cyl=.2, disp=.15, vs=.15) |>
  clin_add_titles(
    list(
      c("Left", "Center", "Right"),
      c("Just the middle")
```

```
    )
  ) |>
  clin_add_footnotes(
      list(
        c(
          "Here's a footnote.",
          format(Sys.time(), "%H:%M %A, %B %d, %Y")
        )
      )
    )

write_clintable(ct, file="example_table.docx")
```



Left

Center
Just the middle

Right

a
1

mpg	cyl	hp	qsec	vs	am
21.0	6	110	16.46	0	1
21.0	6	110	17.02	0	1
22.8	4	93	18.61	1	1
21.4	6	110	19.44	1	0
18.7	8	175	17.02	0	0
18.1	6	105	20.22	1	0
14.3	8	245	15.84	0	0
24.4	4	62	20.00	1	0
22.8	4	95	22.90	1	0
19.2	6	123	18.30	1	0
17.8	6	123	18.90	1	0
16.4	8	180	17.40	0	0
17.3	8	180	17.60	0	0
15.2	8	180	18.00	0	0
10.4	8	205	17.98	0	0
10.4	8	215	17.82	0	0

Here's a footnote.

14:02 Tuesday, March 04, 2025

WHY THIS FRAMEWORK?

So why **{clinify}**? As we explained earlier, the key idea is that {flextable} and {officer} have so much of the functionality that's already needed - so **{clinify}** focuses on specific additional features and streamline certain pieces to make the development of tables more straightforward. One last point is that in building this package, we also didn't want to reinvent the wheel. Several other packages coordinate with the {flextable} and {officer} ecosystem, such as {rtables} or {gtsummary}. Our intent is that **{clinify}** can hopefully work with these packages as well introduce some of this tedious additional functionality.

CONTACT INFORMATION (HEADER 1)

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Mike Stackhouse
Company: Atorus Research
Email: Mike.Stackhouse@atorusresearch.com
Website: www.atorusresearch.com