Tables for presentation

Hospital	Total cases with known outcome	Recovered			To
		Total	% of cases	Median CT values	
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

In this chapter we'll learn how to convert summary data frames into presentation-ready tables with the **flextable** package. These tables can be inserted into powerpoint slides, HTML pages, PDF or Word documents, etc.

Understand that *before* using **flextable**, you must create the summary table as a data frame as we saw in the previous chapter. The resulting data frame can then be passed to **flextable** for display formatting.
There are many other R packages that can be used to craft tables for presentation - we chose to highlight **flextable** in this chapter.

Preparation

Load packages

Install and load **flextable**. In this handbook we emphasize `p_load()` from **pacman**, which installs the package if necessary *and* loads it for use. You can also load packages with `library()` from **base** R. See the page on [R basics] for more information on R packages.

```
pacman::p_load(
  rio,            # import/export
  here,           # file pathways
  flextable,      # make HTML tables
  officer,        # helper functions for tables
```

```
  tidyverse)      # data management, summary, and visualization
```

Import data

To begin, we import the cleaned linelist of cases from a simulated Ebola epidemic. If you want to follow along, click to download the "clean" linelist (as .rds file). Import data with the `import()` function from the **rio** package (it handles many file types like .xlsx, .csv, .rds - see the [Import and export] page for details).

```
# import the linelist
linelist <- import("linelist_cleaned.rds")
```

Prepare table

Before beginning to use **flextable** you will need to *create* your table as a data frame. See the page on [Descriptive tables] and [Pivoting data] to learn how to create a data frame using packages such as **janitor** and **dplyr**. You must arrange the content in rows and columns as you want it displayed. Then, the data frame will be passed to **flextable** to display it with colors, headers, fonts, etc.

Below is an example from the [Descriptive tables] page of converting the case `linelist` into a data frame that summarises patient outcomes and CT values by hospital, with a Totals row at the bottom. The output is saved as `table`.

```
table <- linelist %>%

  # Get summary values per hospital-outcome group
  ################################################
  group_by(hospital, outcome) %>%                    # Group data
  summarise(                                         # Create new summary columns of ind
    N = n(),                                          # Number of rows per hospital-outc
    ct_value = median(ct_blood, na.rm=T)) %>%         # median CT value per group

  # add totals
  #############
  bind_rows(                                         # Bind the previous table with this
    linelist %>%
      filter(!is.na(outcome) & hospital != "Missing") %>%
      group_by(outcome) %>%                          # Grouped only by outcome, not by h
      summarise(
```

```
        N = n(),                                          # Number of rows for whole dataset
        ct_value = median(ct_blood, na.rm=T))) %>%        # Median CT for whole dataset

  # Pivot wider and format
  ########################
  mutate(hospital = replace_na(hospital, "Total")) %>%
  pivot_wider(                                            # Pivot from long to wide
    values_from = c(ct_value, N),                          # new values are from ct and count
    names_from = outcome) %>%                              # new column names are from outcom
  mutate(                                                 # Add new columns
    N_Known = N_Death + N_Recover,                                  # number with known outco
    Pct_Death = scales::percent(N_Death / N_Known, 0.1),            # percent cases who died
    Pct_Recover = scales::percent(N_Recover / N_Known, 0.1)) %>% # percent who recovered (
  select(                                                 # Re-order columns
    hospital, N_Known,                                     # Intro columns
    N_Recover, Pct_Recover, ct_value_Recover,              # Recovered columns
    N_Death, Pct_Death, ct_value_Death)  %>%               # Death columns
  arrange(N_Known)                                       # Arrange rows from lowest to highes

table  # print
```

```
# A tibble: 7 x 8
# Groups:   hospital [7]
  hospital      N_Known N_Recover Pct_Recover ct_value_Recover N_Death Pct_Death
  <chr>           <int>     <int> <chr>                  <dbl>   <int> <chr>
1 St. Mark's M~     325       126 38.8%                     22     199 61.2%
2 Central Hosp~     358       165 46.1%                     22     193 53.9%
3 Other             685       290 42.3%                     21     395 57.7%
4 Military Hos~     708       309 43.6%                     22     399 56.4%
5 Missing          1125       514 45.7%                     21     611 54.3%
6 Port Hospital    1364       579 42.4%                     21     785 57.6%
7 Total            3440      1469 42.7%                     22    1971 57.3%
# i 1 more variable: ct_value_Death <dbl>
```

Basic flextable

Create a flextable

To create and manage **flextable** objects, we first pass the data frame through the `flextable()` function. We save the result as `my_table`.

```
my_table <- flextable(table)
my_table
```

hospital	N_Known	N_Recover	Pct_Recover	ct_value_Recover	N_Death	Pct_Death	ct_value_Death
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	199	61.2%	22
Central Hospital	358	165	46.1%	22	193	53.9%	22
Other	685	290	42.3%	21	395	57.7%	22
Military Hospital	708	309	43.6%	22	399	56.4%	21
Missing	1,125	514	45.7%	21	611	54.3%	21
Port Hospital	1,364	579	42.4%	21	785	57.6%	22
Total	3,440	1,469	42.7%	22	1,971	57.3%	22

After doing this, we can progressively pipe the `my_table` object through more **flextable** formatting functions.

In this page for sake of clarity we will save the table at intermediate steps as `my_table`, adding **flextable** functions bit-by-bit. If you want to see *all* the code from beginning to end written in one chunk, visit the All code together section below.

The general syntax of each line of **flextable** code is as follows:

- `function(table, i = X, j = X, part = "X")`, where:
  - The 'function' can be one of many different functions, such as `width()` to determine column widths, `bg()` to set background colours, `align()` to set whether text is centre/right/left aligned, and so on.
  - `table =` is the name of the data frame, although does not need to be stated if the data frame is piped into the function.
  - `part =` refers to which part of the table the function is being applied to. E.g. "header", "body" or "all".
  - `i =` specifies the *row* to apply the function to, where 'X' is the row number. If multiple rows, e.g. the first to third rows, one can specify: `i = c(1:3)`. Note if 'body' is selected, the first row starts from underneath the header section.

- `j =` specifies the *column* to apply the function to, where 'x' is the column number or name. If multiple columns, e.g. the fifth and sixth, one can specify: `j = c(5,6)`.

You can find the complete list of **flextable** formatting function here or review the documentation by entering `?flextable`.

Column width

We can use the `autofit()` function, which nicely stretches out the table so that each cell only has one row of text. The function `qflextable()` is a convenient shorthand for `flextable()` and `autofit()`.

```
my_table %>% autofit()
```

hospital	N_Known	N_RecoverPct_Recover	ct_value_Recov
St. Mark's Maternity Hospital (SMMH)	325	12638.8%	
Central Hospital	358	16546.1%	
Other	685	29042.3%	
Military Hospital	708	30943.6%	
Missing	1,125	51445.7%	
Port Hospital	1,364	57942.4%	
Total	3,440	1,46942.7%	

However, this might not always be appropriate, especially if there are very long values within cells, meaning the table might not fit on the page.

Instead, we can specify widths with the `width()` function. It can take some playing around to know what width value to put. In the example below, we specify different widths for column 1, column 2, and columns 4 to 8.

```
my_table <- my_table %>%
  width(j=1, width = 2.7) %>%
  width(j=2, width = 1.5) %>%
  width(j=c(4,5,7,8), width = 1)

my_table
```

hospital	N_Known	N_Recover	Pct_Recover	ct_value_Recove	N_I
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	
Central Hospital	358	165	46.1%	22	
Other	685	290	42.3%	21	
Military Hospital	708	309	43.6%	22	
Missing	1,125	514	45.7%	21	
Port Hospital	1,364	579	42.4%	21	
Total	3,440	1,469	42.7%	22	

Column headers

We want more clearer headers for easier interpretation of the table contents.

For this table, we will want to add a second header layer so that columns covering the same subgroups can be grouped together. We do this with the `add_header_row()` function with `top = TRUE`. We provide the new name of each column to `values =`, leaving empty values `""` for columns we know we will merge together later.

We also rename the header names in the now-second header in a separate `set_header_labels()` command.

Finally, to "combine" certain column headers in the top header we use `merge_at()` to merge the column headers in the top header row.

```
my_table <- my_table %>%

  add_header_row(
    top = TRUE,                # New header goes on top of existing header row
    values = c("Hospital",     # Header values for each column below
               "Total cases with known outcome",
               "Recovered",    # This will be the top-level header for this and two next c
               "",
               "",
               "Died",         # This will be the top-level header for this and two next c
               "",             # Leave blank, as it will be merged with "Died"
               "")) %>%

  set_header_labels(         # Rename the columns in original header row
```

```
      hospital = "",
      N_Known = "",
      N_Recover = "Total",
      Pct_Recover = "% of cases",
      ct_value_Recover = "Median CT values",
      N_Death = "Total",
      Pct_Death = "% of cases",
      ct_value_Death = "Median CT values")  %>%

  merge_at(i = 1, j = 3:5, part = "header") %>% # Horizontally merge columns 3 to 5 in new
  merge_at(i = 1, j = 6:8, part = "header")     # Horizontally merge columns 6 to 8 in new

my_table  # print
```

Hospital	Total cases with known outcome	Recovered	
		Total% of cases	Median CT values
St. Mark's Maternity Hospital (SMMH)	325	12638.8%	22
Central Hospital	358	16546.1%	22
Other	685	29042.3%	21
Military Hospital	708	30943.6%	22
Missing	1,125	51445.7%	21
Port Hospital	1,364	57942.4%	21
Total	3,440	1,46942.7%	22

Borders and background

You can adjust the borders, internal lines, etc. with various **flextable** functions. It is often easier to start by removing all existing borders with `border_remove()`.

Then, you can apply default border themes by passing the table to `theme_box()`, `theme_booktabs()`, or `theme_alafoli()`.

You can add vertical and horizontal lines with a variety of functions. `hline()` and `vline()` add lines to a specified row or column, respectively. Within each, you must specify the `part =` as either "all", "body", or "header". For vertical lines, specify the column to `j =`, and

for horizontal lines the row to `i =`. Other functions like `vline_right()`, `vline_left()`, `hline_top()`, and `hline_bottom()` add lines to the outsides only.

In all of these functions, the actual line style itself must be specified to `border =` and must be the output of a separate command using the `fp_border()` function from the **officer** package. This function helps you define the width and color of the line. You can define this above the table commands, as shown below.

```
# define style for border line
border_style = officer::fp_border(color="black", width=1)

# add border lines to table
my_table <- my_table %>%

  # Remove all existing borders
  border_remove() %>%

  # add horizontal lines via a pre-determined theme setting
  theme_booktabs() %>%

  # add vertical lines to separate Recovered and Died sections
  vline(part = "all", j = 2, border = border_style) %>%   # at column 2
  vline(part = "all", j = 5, border = border_style)       # at column 5

my_table
```

Hospital	Total cases with known outcome	Recovered	
		Total% of cases	Median CT values
St. Mark's Maternity Hospital (SMMH)	325	12638.8%	22
Central Hospital	358	16546.1%	22
Other	685	29042.3%	21
Military Hospital	708	30943.6%	22
Missing	1,125	51445.7%	21
Port Hospital	1,364	57942.4%	21
Total	3,440	1,46942.7%	22

Font and alignment

We centre-align all columns aside from the left-most column with the hospital names, using the `align()` function from **flextable**.

```
my_table <- my_table %>%
   flextable::align(align = "center", j = c(2:8), part = "all")
my_table
```

Hospital	Total cases with known outcome	Recovered			
		Total	% of cases	Median CT values	To
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	3,440	1,469	42.7%	22	1,9

Additionally, we can increase the header font size and change then to bold. We can also change the total row to bold.

```
my_table <-  my_table %>%
  fontsize(i = 1, size = 12, part = "header") %>%   # adjust font size of header
  bold(i = 1, bold = TRUE, part = "header") %>%     # adjust bold face of header
  bold(i = 7, bold = TRUE, part = "body")           # adjust bold face of total row (row 7

my_table
```

Hospital	Total cases with known outcome	Recovered			
		Total	% of cases	Median CT values	To
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

We can ensure that the proportion columns display only one decimal place using the function `colformat_num()`. Note this could also have been done at data management stage with the `round()` function.

```
my_table <- colformat_num(my_table, j = c(4,7), digits = 1)
my_table
```

Hospital	Total cases with known outcome	Recovered			
		Total	% of cases	Median CT values	To
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

Merge cells

Just as we merge cells horizontally in the header row, we can also merge cells vertically using `merge_at()` and specifying the rows (`i`) and column (`j`). Here we merge the "Hospital" and "Total cases with known outcome" values vertically to give them more space.

```
my_table <- my_table %>%
  merge_at(i = 1:2, j = 1, part = "header") %>%
  merge_at(i = 1:2, j = 2, part = "header")

my_table
```

Hospital	**Total cases with known outcome**	**Recovered**			
		Total	% of cases	Median CT values	To
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

Background color

To distinguish the content of the table from the headers, we may want to add additional formatting. e.g. changing the background color. In this example we change the table body to gray.

```
my_table <- my_table %>%
    bg(part = "body", bg = "gray95")

my_table
```

Hospital	Total cases with known outcome	Recovered			To
		Total	% of cases	Median CT values	
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

Conditional formatting

We can highlight all values in a column that meet a certain rule, e.g. where more than 55% of cases died. Simply put the criteria to the `i =` or `j =` argument, preceded by a tilde `~`. Reference the column in the data frame, not the display heading values.

```
my_table %>%
  bg(j = 7, i = ~ Pct_Death >= 55, part = "body", bg = "red")
```

Hospital	Total cases with known outcome	Recovered			To
		Total	% of cases	Median CT values	
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

Or, we can highlight the entire row meeting a certain criterion, such as a hospital of interest. To do this we just remove the column (`j`) specification so the criteria apply to all columns.

```
my_table %>%
  bg(., i= ~ hospital == "Military Hospital", part = "body", bg = "#91c293")
```

Hospital	Total cases with known outcome	Recovered			
		Total	% of cases	Median CT values	To
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19
Other	685	290	42.3%	21	39
Military Hospital	708	309	43.6%	22	39
Missing	1,125	514	45.7%	21	61
Port Hospital	1,364	579	42.4%	21	78
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,9**

All code together

Below we show all the code from the above sections together.

```
border_style = officer::fp_border(color="black", width=1)

pacman::p_load(
  rio,            # import/export
  here,           # file pathways
  flextable,      # make HTML tables
  officer,        # helper functions for tables
  tidyverse)      # data management, summary, and visualization

table <- linelist %>%

  # Get summary values per hospital-outcome group
  ###############################################
  group_by(hospital, outcome) %>%                    # Group data
  summarise(                                         # Create new summary columns of ind
```

```
  N = n(),                                          # Number of rows per hospital-outc
  ct_value = median(ct_blood, na.rm=T)) %>%         # median CT value per group

# add totals
#############
bind_rows(                                         # Bind the previous table with this
  linelist %>%
    filter(!is.na(outcome) & hospital != "Missing") %>%
    group_by(outcome) %>%                          # Grouped only by outcome, not by h
    summarise(
      N = n(),                                     # Number of rows for whole dataset
      ct_value = median(ct_blood, na.rm=T))) %>%   # Median CT for whole dataset

# Pivot wider and format
########################
mutate(hospital = replace_na(hospital, "Total")) %>%
pivot_wider(                                       # Pivot from long to wide
  values_from = c(ct_value, N),                    # new values are from ct and count
  names_from = outcome) %>%                        # new column names are from outcom
mutate(                                            # Add new columns
  N_Known = N_Death + N_Recover,                                 # number with known outco
  Pct_Death = scales::percent(N_Death / N_Known, 0.1),           # percent cases who died
  Pct_Recover = scales::percent(N_Recover / N_Known, 0.1)) %>% # percent who recovered (
select(                                            # Re-order columns
  hospital, N_Known,                               # Intro columns
  N_Recover, Pct_Recover, ct_value_Recover,        # Recovered columns
  N_Death, Pct_Death, ct_value_Death)  %>%         # Death columns
arrange(N_Known) %>%                               # Arrange rows from lowest to highe

# formatting
############
flextable() %>%              # table is piped in from above
add_header_row(
  top = TRUE,                # New header goes on top of existing header row
  values = c("Hospital",     # Header values for each column below
             "Total cases with known outcome",
             "Recovered",    # This will be the top-level header for this and two next c
             "",
             "",
             "Died",         # This will be the top-level header for this and two next c
             "",             # Leave blank, as it will be merged with "Died"
```

```
                "")) %>%
    set_header_labels(          # Rename the columns in original header row
      hospital = "",
      N_Known = "",
      N_Recover = "Total",
      Pct_Recover = "% of cases",
      ct_value_Recover = "Median CT values",
      N_Death = "Total",
      Pct_Death = "% of cases",
      ct_value_Death = "Median CT values")  %>%
  merge_at(i = 1, j = 3:5, part = "header") %>% # Horizontally merge columns 3 to 5 in new
  merge_at(i = 1, j = 6:8, part = "header") %>%
  border_remove() %>%
  theme_booktabs() %>%
  vline(part = "all", j = 2, border = border_style) %>%   # at column 2
  vline(part = "all", j = 5, border = border_style) %>%   # at column 5
  merge_at(i = 1:2, j = 1, part = "header") %>%
  merge_at(i = 1:2, j = 2, part = "header") %>%
  width(j=1, width = 2.7) %>%
  width(j=2, width = 1.5) %>%
  width(j=c(4,5,7,8), width = 1) %>%
  flextable::align(., align = "center", j = c(2:8), part = "all") %>%
  bg(., part = "body", bg = "gray95")  %>%
  bg(., j=c(1:8), i= ~ hospital == "Military Hospital", part = "body", bg = "#91c293") %>%
  colformat_num(., j = c(4,7), digits = 1) %>%
  bold(i = 1, bold = TRUE, part = "header") %>%
  bold(i = 7, bold = TRUE, part = "body")
```

```
`summarise()` has grouped output by 'hospital'. You can override using the
`.groups` argument.
```

```
table
```

Hospital	Total cases with known outcome	Recovered			To
		Total	% of cases	Median CT values	
St. Mark's Maternity Hospital (SMMH)	325	126	38.8%	22	19
Central Hospital	358	165	46.1%	22	19

Hospital	Total cases with known outcome	Recovered			
		Total	% of cases	Median CT values	To
Other	685	290	42.3%	21	3
Military Hospital	708	309	43.6%	22	3
Missing	1,125	514	45.7%	21	6
Port Hospital	1,364	579	42.4%	21	7
Total	**3,440**	**1,469**	**42.7%**	**22**	**1,**

Saving your table

There are different ways the table can be integrated into your output.

Save single table

You can export the tables to Word, PowerPoint or HTML or as an image (PNG) files. To do this, use one of the following functions:

- `save_as_docx()`
- `save_as_pptx()`
- `save_as_image()`
- `save_as_html()`

For instance below we save our table as a word document. Note the syntax of the first argument - you can just provide the name of your flextable object e.g. `my_table`, or you can give is a "name" as shown below (the name is "my table"). If name, this will appear as the title of the table in Word. We also demonstrate code to save as PNG image.

```
# Edit the 'my table' as needed for the title of table.
save_as_docx("my table" = my_table, path = "file.docx")

save_as_image(my_table, path = "file.png")
```

Note the packages `webshot` or `webshot2` are required to save a flextable as an image. Images may come out with transparent backgrounds.

If you want to view a 'live' version of the **flextable** output in the intended document format, use `print()` and specify one of the below to `preview =`. The document will "pop-up" open on your computer in the specified software program, but will not be saved. This can be useful to check if the table fits in one page/slide or so you can quickly copy it into another document, you can use the print method with the argument preview set to "pptx" or "docx".

```
print(my_table, preview = "docx") # Word document example
print(my_table, preview = "pptx") # Powerpoint example
```