Introduction to Huxtable

David Hugh-Jones

2021-05-01

Contents

Introduction **2**

About this document . . . 2

Huxtable . . . 2

Installation . . . 2

Getting started **2**

Huxtable properties . . . 3

Selecting rows and columns . . . 6

Editing huxtables **7**

Formatting cell text . . . 7

Borders . . . 8

Changing and adding content . . . 10

Changing the layout **12**

Table position and column width . . . 16

Headers . . . 17

Splitting tables . . . 17

Themes **18**

Conditional formatting **19**

Output to different formats **20**

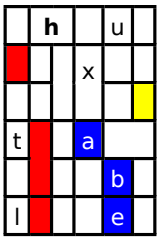
Pretty-printing data frames . . . 20

Using huxtables in knitr and rmarkdown . . . 21

Quick output commands . . . 22

Creating a regression table **22**

Getting more information **23**



Introduction

About this document

This is the introductory vignette for the R package 'huxtable', version 5.3.0. A current version is available on the web in HTML or PDF format.

Huxtable

Huxtable is a package for creating *text tables*. It is powerful, but easy to use. Huxtable's features include:

- Export to LaTeX, HTML, Microsoft Word, Microsoft Excel, Microsoft Powerpoint, RTF and Markdown
- Easy integration with knitr and rmarkdown documents
- Formatted on-screen display
- Multirow and multicolumn cells
- Fine-grained control over cell background, spacing, alignment, size and borders
- Control over text font, style, size, colour, alignment, number format and rotation
- Table manipulation using standard R subsetting, or dplyr functions like `filter` and `select`
- Easy conditional formatting based on table contents
- Quick table themes
- Automatic creation of regression output tables with the `huxreg` function

We will cover many of these features below.

Installation

You can install huxtable from within R:

```
install.packages("huxtable")
```

Getting started

A huxtable is an R object representing a table of text. You already know that R can represent a table of data in a data frame. For example, if `mydata` is a data frame, then `mydata[1, 2]` represents the the data in row 1, column 2.

A huxtable is just a data frame with some extra properties. So, if `myhux` is a huxtable, then `myhux[1, 2]` represents the data in row 1 column 2, as before. But this cell will also have some other properties - for example, the font size of the text, or the colour of the cell border.

To create a huxtable, use the function `huxtable`, or `hux` for short. Let's suppose we want to print a table of jams that we have for sale. There are two columns, representing the kind of jam, and its price:

```
library(huxtable)

jams <- hux(
        Type  = c("Strawberry", "Raspberry", "Plum"),
        Price = c(1.90, 2.10, 1.80)
      )
```

You can convert a data frame to a huxtable with as_hux.

```
data(mtcars)
car_ht <- as_hux(mtcars)
```

If you look at a huxtable in R, it will print out a simple representation of the data. Notice that we've added the column names to the data. We're going to print them out, so they need to be part of the actual table. The data will start on row 2 of the huxtable, and the column names will be row 1.

```
print_screen(jams)     # on the R command line, you can just type "jams"
```

```
##                                      Type             Price
##                                      Strawberry         1.9
##                                      Raspberry          2.1
##                                      Plum               1.8
##
## Column names: Type, Price
```

To print a huxtable as LaTeX or HTML, just call `print_latex` or `print_html`. In knitr documents, like this one, you can simply evaluate the huxtable:

```
jams
```

Type	Price
Strawberry	1.9
Raspberry	2.1
Plum	1.8

Huxtable properties

The default output is a plain table. Let's make it smarter. We'll:

- make the table headings bold;
- draw a border under the first row;
- tweak the table width and spacing;
- change the number formatting;
- and add a caption.

```
library(dplyr)

jams %>%
      set_all_padding(4) %>%
      set_outer_padding(0) %>%
      set_number_format(2) %>%
      set_bold(1, everywhere) %>%
      set_bottom_border(1, everywhere) %>%
      set_width(0.4) %>%
      set_caption("Pots of jam for sale")
```

Table 1: Pots of jam for sale

Type	Price
Strawberry	1.90
Raspberry	2.10
Plum	1.80

All these functions set one or more *properties* on the huxtable. That's why they all start with `set_...`. The functions return the modified huxtable. So you can chain them together using the magrittr pipe. Really, these functions evaluate to:

```
jams <- set_all_padding(jams, 4)
jams <- set_outer_padding(jam, 0)
# etc.
```

Let's go through them line by line.

- `jams %>% set_all_padding(10)` sets four properties on every cell of the huxtable: the `left_padding`, `right_padding`, `top_padding` and `bottom_padding` property. We could have called `set_left_padding(10)` and so on, but this is a convenient shortcut. Cell padding is the amount of space on each side of a table cell. If you're familiar with HTML, you'll know how this works.
- `set_outer_padding(jams, 0)` sets the padding around the outside of the huxtable to 0. Again, this is a shortcut. It's like setting `left_padding` on all the cells on the left side of the huxtable, `top_padding` on the top, and so on.
- `set_number_format(jams, 2)` changes how numbers within cells are displayed. It will work not just on numeric data, but on any numbers found in a cell. Setting the `number_format` property to 2 means that numbers will have 2 decimal places.
- `set_bold(jams, 1, everywhere)` sets the `bold` property. This time we don't set it for all cells – only on cells in row **1** and in all columns, i.e. `everywhere`. `set_bold()` has a default value of `TRUE`, so the call is just short for `set_bold(jams, 1, everywhere, TRUE)`.
- `set_bottom_border(jams, 1, everywhere)` sets the `bottom_border` property. Again it's set for cells in row 1 and all columns. The `bottom_border` property is the width of the border in points. Here, we've set it to its default value of 0.4.
- So far, all these properties have been *cell properties*. The next line sets a *table property* which applies to the whole table: the width.
- The last line, `set_caption(...)`, sets another *table property*: the table caption.

Incidentally, I've used a dplyr style to set these properties, chaining calls together in a pipe. But you don't need to do that. You can also set properties directly. Here's a set of calls that do exactly the same as the above:

```
# set all padding:
left_padding(jams) <- 4
right_padding(jams) <- 4
top_padding(jams) <- 4
bottom_padding(jams) <- 4

# set outer padding:
left_padding(jams)[1:nrow(jams), 1] <- 0
top_padding(jams)[1, 1:ncol(jams)] <- 0
right_padding(jams)[1:nrow(jams), ncol(jams)] <- 0
bottom_padding(jams)[nrow(jams), 1:ncol(jams)] <- 0

number_format(jams) <- 2
bold(jams)[1, 1:ncol(jams)] <- TRUE
bottom_border(jams)[1, 1:ncol(jams)] <- 0.4
width(jams) <- 0.4
caption(jams) <- "Pots of jam for sale"
```

This way of setting properties is the same as using functions like `names(x) <- c("Name 1", "Name 2", ...)` in base R. You can write

```
names(x)[1] <- "Name"
```

to change the first name of a vector. Similarly, in huxtable, you can write

```
bold(jams)[1, 1:ncol(jams)] <- TRUE
```

to set the bold property on the first row of cells.

Here, the assignment style is a little more verbose than the dplyr style, and you don't get convenient shortcuts like `everywhere`. But you can use whichever you prefer.

To sum up, you set *cell properties* on a huxtable like this:

```
ht <- set_property(ht, rows, cols, value)
```

or like this:

```
ht <- set_property(ht, value)
```

where `property` is the name of the huxtable property. The first form sets the cell property for specific rows and columns. The second form sets it for all cells. *Table-level properties* are always set like

```
ht <- set_property(ht, value)
```

since they always apply to the whole table.

As well as cell properties and table properties, there are also row properties and column properties. The table below shows a complete list of properties.

Table 2: Huxtable properties

Cell Text	Cell	Row	Column	Table
bold	align	row_height	col_width	caption
escape_contents	background_color	header_rows	header_cols	caption_pos
font	bottom_border			caption_width
font_size	bottom_border_color			height
italic	bottom_border_style			label
markdown	bottom_padding			latex_float
na_string	colspan			position
number_format	left_border			table_environment
rotation	left_border_color			tabular_environment
text_color	left_border_style			width
wrap	left_padding			
	right_border			
	right_border_color			
	right_border_style			
	right_padding			
	rowspan			
	top_border			
	top_border_color			
	top_border_style			
	top_padding			
	valign			

Selecting rows and columns

When you call `set_property(ht, rows, cols, value)`, you can specify `rows` and `cols` in several different ways.

- You can use numbers:

```
# Set the italic property on row 1, column 1:
jams %>% set_italic(1, 1)
```

- Or use logical indices:

```
# Set the italic property on column 1 of every row matching "berry":
is_berry <- grepl("berry", jams$Type)
jams %>% set_italic(is_berry, 1)
```

- Or use characters for column names:

```
# Set the italic property on row 1 of the column named "Type":
jams %>% set_italic(1, "Type")
```

These methods should all be familiar from base R. They are just the same as you can use for subsetting a data frame. In fact, you can use the same methods for assignment style:

```
italic(jams)[1, "Type"] <- TRUE
# the same as:
jams <- jams %>% set_italic(1, "Type")
```

In `set_` functions, there are some extra methods:

- You can use tidyselect functions like `matches()` or `starts_with()` to select columns:

```
# Set the italic property on row 1 of every column whose name starts with "T":
jams %>%
      set_italic(1, starts_with("T"))
```

There are also some huxtable-specific selectors.

- `everywhere` sets a property on all rows, or all columns.

```
# Set the italic property on row 1 of all columns:
jams %>% set_italic(1, everywhere)

# Set the italic property on all rows of column 1:
jams %>% set_italic(everywhere, 1)
```

- `final(n)` sets a property on the last `n` rows or columns.

```
jams %>% set_italic(final(2), everywhere)
# same as:
jams %>% set_italic(3:4, 1:2)
```

Editing huxtables

Formatting cell text

Here are some useful ways to change how cells are displayed.

- The `bold` property makes a whole cell bold, and the `italic` property makes a cell italic. We've seen these.
- The `text_color` property changes the color of text.

```
jams %>%
      set_text_color(2:3, 1, "purple")
```

Table 3: Pots of jam for sale

Type	Price
Strawberry	1.90
Raspberry	2.10
Plum	1.80

You can use any valid R color name, or an HTML hex color like `#FF0000`.

- The `background_color` property changes background color.

  Here's one way to apply a subtle horizontal stripe to a table:

```
jams %>%
      set_background_color(evens, everywhere, "grey95")
```

Table 4: Pots of jam for sale

Type	Price
Strawberry	1.90
Raspberry	2.10
Plum	1.80

This uses another huxtable-specific shortcut: `evens` specifies even-numbered rows or columns. (And `odds` specifies odd-numbered rows or columns.)

- If you want to format selected text within cells, you can use markdown by setting the `markdown` property.

  The `set_markdown_contents()` sets the markdown property and the cell contents together:

```
jams %>%
      set_markdown_contents(1, 1, "*Type* of jam") %>%
      set_markdown_contents(1, 2, "*Price* of jam") %>%
      set_markdown_contents(3, 2, "~~2.10~~ **Sale!** 1.50")
```

Table 5: Pots of jam for sale

***Type* of jam**	***Price* of jam**
Strawberry	1.90
Raspberry	~~2.10~~ **Sale!** 1.50
Plum	1.80

Borders

Each huxtable cell has 4 borders, on the left, top, right and bottom. These borders are "collapsed", in CSS parlance: row 1's bottom border is row 2's top border, and setting one automatically sets the other. Each border has a thickness, a style ("solid", "double", "dotted" or "dashed") and a colour.

To set all these properties together, you can use a `brdr()` object:

```
jams %>%
      set_right_border(everywhere, 1, brdr(3, "double", "grey"))
```

Table 6: Pots of jam for sale

Type	Price
Strawberry	1.90
Raspberry	2.10
Plum	1.80

Or, you can set each component individually:

```
jams %>%
      set_right_border(everywhere, 1, 3) %>%
      set_right_border_style(everywhere, 1, "double") %>%
      set_right_border_color(everywhere, 1, "grey")
```

Table 7: Pots of jam for sale

Type	Price
Strawberry	1.90
Raspberry	2.10
Plum	1.80

To set all the borders around a cell, use `set_all_borders()`. Here's a corporate look for our jams:

```
jams %>%
      set_background_color(evens, everywhere, "grey80") %>%
      set_background_color(odds, everywhere, "grey90") %>%
      set_all_borders(brdr(0.4, "solid", "white")) %>%
      set_outer_padding(4)
```

Table 8: Pots of jam for sale

Type	Price
Strawberry	1.90
Raspberry	2.10
Plum	1.80

Other shortcuts include:

- `set_tb_borders()` to set top and bottom borders;
- `set_lr_borders()` to set left and right borders;
- `set_outer_borders()` to set borders around a group of cells.

Not all output formats handle all kinds of borders equally well. In particular, LaTeX currently only handles "solid" and "double" borders – not "dotted" or "dashed".

Changing and adding content

You can treat a huxtable just like a data frame. For example, here's how to change the text in a particular cell:

```
jams[3, 1] <- "Snozberry"
```

You can change a whole column like this:

```
# Summer sale!
jams$Price <- c("Price", 1.50, 1.60, 1.50)
```

Notice that since the "Price" label is part of the huxtable, I had to include it in the data.

Or you can add a new column the same way.

```
jams$Sugar <- c("Sugar content", "40%", "50%", "30%")
jams
```

Table 9: Pots of jam for sale

Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	50.00%
Plum	1.80	30.00%

Notice that the new column has the same bold heading, borders and number formatting as the other two. When you add data to a huxtable, by default, it copies cell properties over from the nearest neighbour.

Similarly, you can add a new row to a huxtable with `rbind`, and cell properties will be copied from the previous row:

```
rbind(jams, c("Gooseberry", 2.1, "55%"))
```

Table 10: Pots of jam for sale

Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	50.00%
Plum	1.80	30.00%
Gooseberry	2.10	55.00%

Sometimes, you would like to insert rows or columns in the middle of a table. You can do this with rbind, but it is not very convenient:

```
best_before <- c("Best before", c("Aug 2022", "Sept 2022", "June 2022"))

cbind(jams[, 1], best_before, jams[, -1])
```

Table 11: Pots of jam for sale

Type	Best before	Price	Sugar content
Strawberry	Aug 2022.00	1.90	40.00%
Raspberry	Sept 2022.00	2.10	50.00%
Plum	June 2022.00	1.80	30.00%

Huxtable has a useful shortcut called `insert_column()` for this.

```
jams %>%
      insert_column(best_before, after = "Type") %>%
      set_number_format(everywhere, 2, 0) # correct the formatting for dates
```

Table 12: Pots of jam for sale

Type	Best before	Price	Sugar content
Strawberry	Aug 2022	1.90	40.00%
Raspberry	Sept 2022	2.10	50.00%
Plum	June 2022	1.80	30.00%

The `after` argument says where the second object should be inserted. It can be a column name or number. There's also an `insert_row()` function.

If you prefer using dplyr to edit contents, many dplyr functions work with huxtable.

```
jams %>%
      mutate(
        Type = toupper(Type)
      ) %>%
      select(Type, Price)
```

Table 13: Pots of jam for sale

TYPE	Price
STRAWBERRY	1.90
RASPBERRY	2.10
PLUM	1.80

Notice that changing the `Type` column changed the whole column, including the heading. If you want to work with the underlying data, it's often best to do this before creating a huxtable. For example, here's how you might create a `jams` table ordered by price:

```
jams_data <- data.frame(
        Type = c("Strawberry", "Raspberry", "Plum"),
        Price = c(1.90, 2.10, 1.80)
      )

jams_ordered <- jams_data %>%
      arrange(Price) %>%
      as_hux() %>%
      set_bold(1, everywhere) # et cetera...
```

It's easier to arrange by `Price` before you add the "Price" heading to the column. Alternatively, you can use `as_hux(..., add_colnames = FALSE)`, and add column names later with the `add_colnames()` function.

```
# Same result as above

jams_data %>%
      as_hux(add_colnames = FALSE) %>%
      arrange(Price) %>%
      add_colnames()
```

Changing the layout

When we have larger tables, we may need to control the layout more carefully. Here's selected rows of the `iris` dataset:

```
iris_hux <- iris %>%
      group_by(Species) %>%
      select(Species, Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) %>%
      slice(1:5) %>%
      as_hux() %>%
      theme_basic() %>%
      set_tb_padding(2)

iris_hux
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5	3.6	1.4	0.2
versicolor	7	3.2	4.7	1.4
versicolor	6.4	3.2	4.5	1.5
versicolor	6.9	3.1	4.9	1.5
versicolor	5.5	2.3	4	1.3
versicolor	6.5	2.8	4.6	1.5
virginica	6.3	3.3	6	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3	5.8	2.2

Here I've used `theme_basic()` to quickly provide an acceptable look. We'll see more about themes later.

The column names are rather long. We could use an extra header row to shorten them.

```
iris_hux <- iris_hux %>%
  set_contents(1, 2:5, c("Length", "Width", "Length", "Width")) %>%
  insert_row("", "Sepal", "", "Petal", "", after = 0) %>%
  merge_cells(1, 2:3) %>%
  merge_cells(1, 4:5) %>%
  set_align(1, everywhere, "center") %>%
  set_tb_padding(1, everywhere, 0) %>%
  set_bold(1, everywhere)

iris_hux
```

	Sepal		Petal	
Species	**Length**	**Width**	**Length**	**Width**
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5	3.6	1.4	0.2
versicolor	7	3.2	4.7	1.4
versicolor	6.4	3.2	4.5	1.5
versicolor	6.9	3.1	4.9	1.5
versicolor	5.5	2.3	4	1.3
versicolor	6.5	2.8	4.6	1.5
virginica	6.3	3.3	6	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3	5.8	2.2

Let's take this piece by piece.

- `set_contents()` is a shortcut to change contents, for use within pipes. It's equivalent to saying `iris_hux[1, 2:5] <- c("Length", ...)`.
- `insert_row()` inserts a new row at the top.
- `merge_cells(1, 2:3)` merges the cells in row 1, columns 2 and 3. These now become a single cell. If you know HTML, this is equivalent to setting the `colspan` of column 2 to 2.
- `merge_cells(1, 4:5)` does the same for row 1, columns 4 and 5.
- Lastly, `set_align()` centres all the cells in the first row and `set_tb_padding()` fixes up the vertical padding, to keep these cells close to the row below.

This looks better, but it is rather long. (And we only used a few of the 150 rows in the `iris` data!) One solution is to reorganize your table layout. In data management, it is a cardinal sin to have the same data in two columns, but it can make a table easier to read.

```
iris_hux_wide <- iris_hux %>%
      set_header_rows(1:2, TRUE) %>%
      restack_across(rows = 7) %>%
      set_bottom_border(final(1), everywhere)

iris_hux_wide
```

	Sepal		Petal			Sepal		Petal			Sepal		Petal	
ecies	Length	Width	Length	Width	Species	Length	Width	Length	Width	Species	Length	Width	Length	Wi
osa	5.1	3.5	1.4	0.2	versicolor	7	3.2	4.7	1.4	virginica	6.3	3.3	6	
osa	4.9	3	1.4	0.2	versicolor	6.4	3.2	4.5	1.5	virginica	5.8	2.7	5.1	
osa	4.7	3.2	1.3	0.2	versicolor	6.9	3.1	4.9	1.5	virginica	7.1	3	5.9	
osa	4.6	3.1	1.5	0.2	versicolor	5.5	2.3	4	1.3	virginica	6.3	2.9	5.6	
osa	5	3.6	1.4	0.2	versicolor	6.5	2.8	4.6	1.5	virginica	6.5	3	5.8	

This is too wide, but we'll deal with that in a second. The `restack_across()` function reorganizes our table to fit into fewer rows (and more columns). There's a similar `restack_down()` function which fits a table into more rows and fewer columns. To understand these, a bit of color will help:

```
lego_hux <- as_hux(matrix(1:16, 4, 4)) %>%
      set_background_color(1:2, 1:2, "red") %>%
      set_background_color(1:2, 3:4, "yellow") %>%
      set_background_color(3:4, 1:2, "darkgreen") %>%
      set_background_color(3:4, 3:4, "blue") %>%
      set_text_color(3:4, 1:4, "white") %>%
      set_all_borders(brdr(2, "solid", "white"))

lego_hux %>% set_caption("Original table")
```

Table 14: Original table

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

```
lego_hux %>%
      restack_across(rows = 2) %>%
      set_caption("Restacked across")
```

Table 15: Restacked across

1	5	9	13	3	7	11	15
2	6	10	14	4	8	12	16

```
lego_hux %>%
      restack_down(cols = 2) %>%
      set_caption("Restacked down")
```

Table 16: Restacked down

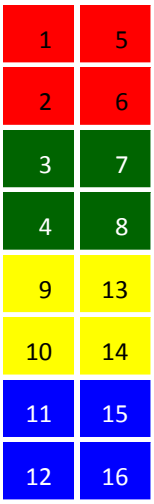


Table position and column width

Our new `iris` huxtable is now shorter, but it's too wide. We can control this with the table-level `width` property. We can also set the width of individual columns with the *column* property `col_width`. And we might want to have this table left-aligned on the page, using the `position` property.

```
iris_hux_wide %>%
      set_width(0.8) %>%
      set_font_size(8) %>%
      set_lr_padding(2) %>%
      set_col_width(rep(c(0.4, 0.2, 0.2, 0.2, 0.2), 3)/3) %>%
      set_position("left")
```

	Sepal		Petal			Sepal		Petal			Sepal		Petal	
Species	**Length**	**Width**	**Length**	**Width**	**Species**	**Length**	**Width**	**Length**	**Width**	**Species**	**Length**	**Width**	**Length**	**Width**
setosa	5.1	3.5	1.4	0.2	versicolor	7	3.2	4.7	1.4	virginica	6.3	3.3	6	2.5
setosa	4.9	3	1.4	0.2	versicolor	6.4	3.2	4.5	1.5	virginica	5.8	2.7	5.1	1.9
setosa	4.7	3.2	1.3	0.2	versicolor	6.9	3.1	4.9	1.5	virginica	7.1	3	5.9	2.1
setosa	4.6	3.1	1.5	0.2	versicolor	5.5	2.3	4	1.3	virginica	6.3	2.9	5.6	1.8
setosa	5	3.6	1.4	0.2	versicolor	6.5	2.8	4.6	1.5	virginica	6.5	3	5.8	2.2

`width` and `col_width` can either be numbers, or units recognized by HTML or LaTeX. It's best to specify `col_width` as a set of numbers. These are treated as proportions of the total table width.

If you have a small table, you may want your text to wrap around it. You can do this by specifying `"wrapleft"` or `"wrapright"` as the position. The table on the right uses `set_position("wrapright")`, `set_width(0.35)` and the "compact" theme, which minimizes cell padding to keep the table small. Table wrapping works in both HTML and LaTeX. There's no option to have text wrapped around both sides of the table. That would just be painful for your readers.

Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	50.00%
Plum	1.80	30.00%

Headers

You'll notice that the restacked `iris` huxtable repeated the header rows appropriately. For this to happen, we set the `header_rows` property to `TRUE` on rows 1-2. This is a *row property*. Row properties are set like:

```
set_row_property(ht, row, value)
```

By themselves, header rows are not displayed any differently. But certain themes will display them differently. You can also style headers yourself using the `style_headers()` function:

```
iris_hux <- iris_hux %>%
      set_header_rows(1:2, TRUE) %>%
      set_header_cols(1, TRUE) %>%
      style_headers(bold = TRUE, text_color = "grey40")

iris_hux
```

	Sepal		Petal	
Species	Length	Width	Length	Width
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5	3.6	1.4	0.2
versicolor	7	3.2	4.7	1.4
versicolor	6.4	3.2	4.5	1.5
versicolor	6.9	3.1	4.9	1.5
versicolor	5.5	2.3	4	1.3
versicolor	6.5	2.8	4.6	1.5
virginica	6.3	3.3	6	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3	5.8	2.2

Here we have set the first two rows as headers, and the first column as a a header column. `style_headers()` applies to both rows and columns. Alternatively, use `style_header_rows()` and `style_header_cols()` to treat header rows and columns differently. Their arguments are a list of properties and property values.

Splitting tables

If we haven't got room to restack, an alternative approach is to split our original table into separate tables. We can do this with `split_across()` and `split_down()`. These functions take a single huxtable and return a list of huxtables. Like the `restack` functions, they take account of headers by default.

```
list_of_iris <- split_across(iris_hux, c(7, 12))
list_of_iris[[1]]
```

	Sepal		Petal	
Species	Length	Width	Length	Width
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5	3.6	1.4	0.2

```
list_of_iris[[2]]
```

	Sepal		Petal	
Species	Length	Width	Length	Width
versicolor	7	3.2	4.7	1.4
versicolor	6.4	3.2	4.5	1.5
versicolor	6.9	3.1	4.9	1.5
versicolor	5.5	2.3	4	1.3
versicolor	6.5	2.8	4.6	1.5

```
list_of_iris[[3]]
```

	Sepal		Petal	
Species	Length	Width	Length	Width
virginica	6.3	3.3	6	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3	5.8	2.2

Themes

Huxtable comes with some predefined themes for formatting. The table of huxtable properties above used `theme_bright()`. Other options include `theme_basic()` and the randomized `theme_mondrian()`:

```
theme_mondrian(jams)
```

Table 17: Pots of jam for sale

Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	50.00%
Plum	1.80	30.00%

The "themes" vignette shows all the available themes. Themes simply apply a set of styles to the huxtable.

Conditional formatting

When you want to apply different formatting to different cells, you can use *mapping functions*.

For example, here's another way to create a striped table:

```
jams %>% map_background_color(by_rows("grey90", "grey95"))
```

Table 18: Pots of jam for sale

Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	50.00%
Plum	1.80	30.00%

Or, we could apply a text color to our `iris` data to pick out the lowest and highest values of each column:

```
iris_hux %>%
      map_text_color(-(1:2), -1,
        by_colorspace("darkred", "grey50", "darkgreen", colwise = TRUE)
      )
```

	Sepal		Petal	
Species	Length	Width	Length	Width
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5	3.6	1.4	0.2
versicolor	7	3.2	4.7	1.4
versicolor	6.4	3.2	4.5	1.5
versicolor	6.9	3.1	4.9	1.5
versicolor	5.5	2.3	4	1.3
versicolor	6.5	2.8	4.6	1.5
virginica	6.3	3.3	6	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3	5.8	2.2

`by_rows` and `by_ranges` are *mapping functions*.

- `by_rows` applies different properties to different rows in sequence.
- `by_colorspace` takes cell numbers as input and maps them to colors.

To use a mapping function, you write `map_property(ht, row, col, fn)`, where `property` is the cell property you want to map. `ht` is the huxtable, and `fn` is the mapping function starting with `by`. `row` and `col` are optional row and column specifiers, just the same as for `set_xxx`.

Here's one more example. To set properties for cells that match a string, use the `by_regex` function.

```
jams %>% map_text_color(by_regex("berry" = "red4", "navy"))
```

Table 19: Pots of jam for sale

Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	50.00%
Plum	1.80	30.00%

There is more information about mapping functions in this article.

Output to different formats

Pretty-printing data frames

If you load `huxtable` within a knitr document, it will automatically format data frames for you:

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

If you don't want this, you can turn it off by setting the `huxtable.knit_print_df` option:

```
options(huxtable.knit_print_df = FALSE)

head(iris) # back to normal
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Using huxtables in knitr and rmarkdown

If you use knitr and rmarkdown in RStudio, huxtable objects should automatically display in the appropriate format (HTML, LaTeX, or RTF).

Huxtable needs some LaTeX packages for LaTeX output. The function `report_latex_dependencies()` will print out a set of `usepackage{...}` statements. If you use Sweave or knitr without rmarkdown, you can use this function in your LaTeX preamble, to load the packages you need.

If you want to create Word or Powerpoint documents, install the flextable package from CRAN. Huxtables can then be automatically printed in Word documents. Or you can convert them to `flextable` objects and include them in Word or Powerpoint documents. Similarly, to print tables in an Excel spreadsheet, install the openxlsx package See `?as_flextable` and `?as_workbook` for more details.

You can print a huxtable on screen by typing its name at the command line. Borders, column and row spans and cell alignment are shown. If the crayon package is installed, and your terminal or R IDE supports it, border, text and background colours are also displayed.

```
print_screen(jams)
```

```
##                                  Pots of jam for sale
##                                Type          Price   Sugar content
##                               ──────────────────────────────────────
##                                Strawberry    1.90          40.00%
##                                Raspberry     2.10          50.00%
##                                Plum          1.80          30.00%
##
## Column names: Type, Price, Sugar
```

If you need to output to another format, file an issue request on Github.

Quick output commands

Sometimes you quickly want to get your data into a document. To do this you can use huxtable functions starting with `quick_`:

- `quick_pdf()` creates a PDF.
- `quick_docx()` creates a Word document.
- `quick_html()` creates a HTML web page.
- `quick_xlsx()` creates an Excel spreadsheet.
- `quick_pptx()` creates a Powerpoint presentation.
- `quick_rtf()` creates an RTF document.
- `quick_latex()` creates a LaTeX file.

These are called with one or more huxtable objects (or objects which can be turned into a huxtable, such as data frames). A new document of the appropriate type will be created and opened. By default the file will be in the current directory, under a name like e.g. `huxtable-output.pdf`. If the file already exists, you'll be asked for confirmation.

```
quick_pdf(iris_hux)
quick_pdf(iris_hux, file = "iris.pdf")
```

Creating a regression table

A common reason to print a table is to report statistical results. The `huxreg()` function creates a table from a set of regressions.

```
lm1 <- lm(mpg ~ cyl, mtcars)
lm2 <- lm(mpg ~ hp, mtcars)
lm3 <- lm(mpg ~ cyl + hp, mtcars)

huxreg(lm1, lm2, lm3)
```

	(1)	(2)	(3)
(Intercept)	37.885 ***	30.099 ***	36.908 ***
	(2.074)	(1.634)	(2.191)
cyl	-2.876 ***		-2.265 ***
	(0.322)		(0.576)
hp		-0.068 ***	-0.019
		(0.010)	(0.015)
N	32	32	32
R2	0.726	0.602	0.741
logLik	-81.653	-87.619	-80.781
AIC	169.306	181.239	169.562

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

For more information see the "huxreg" vignette.

Getting more information

Huxtable has a complete set of help files. These are installed with the package, or readable online.

If you run into trouble, consult `?"huxtable-FAQ"`. It will help you to file a useful bug report or seek help. The NEWS file lists changes in recent versions. The huxtable website has links to all this information and more.