Introduction to Huxtable

David Hugh-Jones

2018-06-02

Contents

Introduction **2**
About this document . . . 2
Huxtable . . . 2
Installation . . . 2
Getting started . . . 2

Changing the look and feel **3**
Huxtable properties . . . 3
Tidyverse syntax . . . 4
Getting properties . . . 5

Editing content **6**
Standard subsetting . . . 6
Using dplyr with huxtable . . . 6
Functions to insert rows, columns and footnotes . . . 7

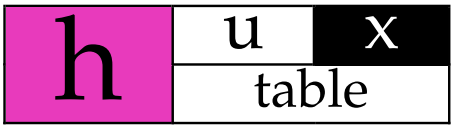
More formatting **7**
Number format . . . 7
Automatic formatting . . . 8
Escaping HTML or LaTeX . . . 9
Width and cell wrapping . . . 9
Adding row and column names . . . 10
Column and row spans . . . 11
Quick themes . . . 12

Selecting rows, columns and cells **13**
Row and column functions . . . 13
Conditional formatting . . . 14

Creating a regression table **16**

Output to different formats **16**
Automatic pretty-printing of data frames . . . 16
Using huxtables in knitr and rmarkdown . . . 17
Quick output commands . . . 18

End matter **18**



Introduction

About this document

This is the introductory vignette for the R package 'huxtable', version 3.0.0.9000. 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. It is meant to be a replacement for packages like xtable, which is useful but not always very user-friendly. Huxtable's features include:

- Export to LaTeX, HTML, Word and Markdown
- Easy integration with knitr and rmarkdown documents
- 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 all of these features below.

Installation

If you haven't already installed huxtable, you can do so from the R command line:

```
install.packages('huxtable')
```

Getting started

A huxtable is a way of representing a table of text data in R. 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, and `mydata$start_time` is all the data in the column called `start_time`.

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 table with huxtable, use the function `huxtable`, or `hux` for short. This works very much like `data.frame`.

```
library(huxtable)
ht <- hux(
        Employee     = c('John Smith', 'Jane Doe', 'David Hugh-Jones'),
        Salary       = c(50000, 50000, 40000),
        add_colnames = TRUE
      )
```

If you already have your data in a data frame, you can convert it 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 frame itself, using the `add_colnames` argument to `hux`. We're going to print them out, so they need to be part of the actual table. **NB:** This means that row 1 of your data will be row 2 of the huxtable, and the column names of your data will be the new row 1.

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

```
##   Employee            Salary
##   John Smith           5e+04
##   Jane Doe             5e+04
##   David Hugh-Jones     4e+04
##
## Column names: Employee, Salary
```

To print a huxtable out using LaTeX or HTML, just call `print_latex` or `print_html`. In knitr documents, like this one, you can simply evaluate the hux. It will know what format to print itself in.

```
ht
```

Employee	Salary
John Smith	5e+04
Jane Doe	5e+04
David Hugh-Jones	4e+04

Changing the look and feel

Huxtable properties

The default output is a very plain table. Let's make it a bit smarter. We'll make the table headings bold, draw a line under the header row, and add some horizontal space to the cells. We also need to change that default number formatting to look less scientific.

To do this, we need to set cell level properties. You set properties by assigning to the property name, just as you assign `names(x) <- new_names` in base R. The following commands assign the value 10 to the `right_padding` and `left_padding` properties, for all cells in `ht`:

```
right_padding(ht) <- 10
left_padding(ht)  <- 10
```

Similarly, we can set the `number_format` property to change how numbers are displayed in cells:

```
number_format(ht) <- 2    # 2 decimal places
```

To assign properties to just some cells, you use subsetting, just as in base R. So, to make the first row of the table **bold** and give it a bottom border, we do:

```
bold(ht)[1, ]          <- TRUE
bottom_border(ht)[1, ] <- 1
```

After these changes, our table looks smarter:

```
ht
```

Employee	**Salary**
John Smith	50000.00
Jane Doe	50000.00
David Hugh-Jones	40000.00

So far, all these properties have been set at cell level. Different cells can have different alignment, text formatting and so on. By contrast, `caption` is a table-level property. It only takes one value, which sets a table caption.

```
caption(ht) <- 'Employee table'
ht
```

Table 1: Employee table

Employee	Salary
John Smith	50000.00
Jane Doe	50000.00
David Hugh-Jones	40000.00

As well as cell properties and table properties, there is also one row property, row heights, and one column property, column widths.

The table below shows a complete list of properties. Most properties work the same for LaTeX and HTML, though there are some exceptions.

Table 2: Huxtable properties

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

Tidyverse syntax

If you prefer a tidyverse style of code, using the pipe operator `%>%`, then you can use `set_*` functions. These have the same name as the property, with `set_` prepended. For example, to set the `bold` property, you use the `set_bold` function.

`set_*` functions return the modified huxtable, so you can chain them together like this:

```
library(dplyr)
hux(
        Employee     = c('John Smith', 'Jane Doe', 'David Hugh-Jones'),
        Salary       = c(50000, 50000, 40000),
        add_colnames = TRUE
      )                                 %>%
```

```
      set_bold(1, 1:2, TRUE)           %>%
      set_bottom_border(1, 1:2, 1)     %>%
      set_align(1:4, 2, 'right')       %>%
      set_right_padding(10)            %>%
      set_left_padding(10)             %>%
      set_caption('Employee table')
```

Table 3: Employee table

Employee	Salary
John Smith	5e+04
Jane Doe	5e+04
David Hugh-Jones	4e+04

`set_*` functions for cell properties are called like this: `set_xxx(ht, row, col, value)` or like this: `set_xxx(ht, value)`. If you use the second form, then the value is set for all cells. `set_*` functions for table properties are always called like `set_xxx(ht, value)`. We'll learn more about this interface in a moment.

There are also four useful convenience functions:

- `set_all_borders` sets left, right, top and bottom borders for selected cells;
- `set_all_border_colors` sets left, right, top and bottom border colors;
- `set_all_padding` sets left, right, top and bottom padding (the amount of space between the content and the border);
- `set_outer_borders` sets an outer border around a rectangle of cells.

Getting properties

To get the current properties of a huxtable, just use the properties function without the left arrow:

```
italic(ht)
```

```
##   Employee Salary
## 1    FALSE  FALSE
## 2    FALSE  FALSE
## 3    FALSE  FALSE
## 4    FALSE  FALSE
```

```
position(ht)
```

```
## [1] "center"
```

As before, you can use subsetting to get particular rows or columns:

```
bottom_border(ht)[1:2,]
```

```
##   Employee Salary
## 1        1      1
## 2        0      0
```

```
bold(ht)[,'Salary']
```

```
##     1     2     3     4
##  TRUE FALSE FALSE FALSE
```

Editing content

Standard subsetting

You can subset, sort and generally data-wrangle a huxtable just like a normal data frame. Cell and table properties will be carried over into subsets.

```
# Select columns by name:
cars_mpg <- car_ht[, c('mpg', 'cyl', 'am')]
# Order by number of cylinders:
cars_mpg <- cars_mpg[order(cars_mpg$cyl),]

cars_mpg <- cars_mpg                              %>%
      huxtable::add_rownames(colname = 'Car') %>%
      huxtable::add_colnames()

cars_mpg[1:5,]
```

Car	mpg	cyl	am
Datsun 710	22.8	4	1
Merc 240D	24.4	4	0
Merc 230	22.8	4	0
Fiat 128	32.4	4	1

Using dplyr with huxtable

You can also use `dplyr` functions to edit a huxtable:

```
car_ht <- car_ht                                                %>%
      huxtable::add_rownames(colname = 'Car')                   %>%
      slice(1:10)                                               %>%
      select(Car, mpg, cyl, hp)                                 %>%
      arrange(hp)                                               %>%
      filter(cyl > 4)                                           %>%
      rename(MPG = mpg, Cylinders = cyl, Horsepower = hp) %>%
      mutate(kml = MPG/2.82)


car_ht <- car_ht                                    %>%
      set_number_format(1:7, 'kml', 2)              %>%
      set_col_width(c(.35, .15, .15, .15, .2)) %>%
      set_width(.6)                                 %>%
      huxtable::add_colnames()

car_ht
```

Car	MPG	Cylinders	Horsepower	kml
Valiant	18.1	6	105	6.42
Mazda RX4	21	6	110	7.45
Mazda RX4 Wag	21	6	110	7.45
Hornet 4 Drive	21.4	6	110	7.59
Merc 280	19.2	6	123	6.81
Hornet Sportabout	18.7	8	175	6.63
Duster 360	14.3	8	245	5.07

In general it is a good idea to prepare your data first, before styling it. For example, it was easier to sort the `cars_mpg` data by cylinder, before adding column names to the data frame itself.

Functions to insert rows, columns and footnotes

Huxtable has three convenience functions for adding a row or column to your table: `insert_row`, `insert_column` and `add_footnote`. `insert_row` and `insert_column` let you add a single row or column. The `after` parameter specifies where in the table to do the insertion, i.e. after what row or column number. `add_footnote` adds a single cell in a new row at the bottom. The cell spans the whole table row, and has a border above.

```
ht <- insert_row(ht, 'Hadley Wickham', '100000', after = 3)
ht <- add_footnote(ht, 'DHJ deserves a pay rise')
ht
```

Table 4: Employee table

Employee	**Salary**
John Smith	50000.00
Jane Doe	50000.00
Hadley Wickham	100000.00
David Hugh-Jones	40000.00

DHJ deserves a pay rise

More formatting

Number format

You can change how huxtable formats numbers using `number_format`. Set `number_format` to a number of decimal places (for more advanced options, see the help files). This affects all numbers, or number-like substrings within your cells.

```
pointy_ht <- hux(c('Do not pad this.', 11.003, 300, 12.02, '12.1 **')) %>% set_all_borders(1)

number_format(pointy_ht) <- 3
pointy_ht
```

Do not pad this.
11.003
300.000
12.020
12.100 **

You can also align columns by decimal places. If you want to do this for a cell, just set the `align` property to '.' (or whatever you use for a decimal point).

```
align(pointy_ht)[2:5, ] <- '.' # not the first row
pointy_ht
```

Do not pad this.
11.003
300.000
12.020
12.100 **

There is currently no true way to align cells by the decimal point in HTML, and only limited possibilities in TeX, so this works by right-padding cells with spaces. The output may look better if you use a fixed width font.

Automatic formatting

By default, when you create a huxtable using `huxtable` or `as_huxtable`, the package will guess defaults for number formatting and alignment, based on the type of data in your columns. Numeric data will be right-aligned or aligned on the decimal point; character data will be left aligned; and the package will try to set sensible defaults for number formatting. If you want to, you can turn this off with `autoformat = FALSE`:

```
my_data <- data.frame(
        Employee           = c('John Smith', 'Jane Doe', 'David Hugh-Jones'),
        Salary             = c(50000L, 50000L, 40000L),
        Performance_rating = c(8.9, 9.2, 7.8)
      )
as_huxtable(my_data, add_colnames = TRUE) # with automatic formatting
```

Employee	Salary	Performance_rating
John Smith	50000	8.9
Jane Doe	50000	9.2
David Hugh-Jones	40000	7.8

```
as_huxtable(my_data, add_colnames = TRUE, autoformat = FALSE) # no automatic formatting
```

Employee	Salary	Performance_rating
John Smith	5e+04	8.9
Jane Doe	5e+04	9.2
David Hugh-Jones	4e+04	7.8

Escaping HTML or LaTeX

By default, HTML or LaTeX code will be escaped:

```
code_ht <- if (is_latex) hux(c('Some maths', '$a^b$')) else
      hux(c('Copyright symbol', '&copy;'))
code_ht
```

Some maths
a^b

To avoid this, set the `escape_contents` property to `FALSE`.

```
escape_contents(code_ht)[2, 1] <- FALSE
code_ht
```

Some maths
a^b

Width and cell wrapping

You can set table widths using the `width` property, and column widths using the `col_width` property. If you use numbers for these, they will be interpreted as proportions of the table width (or for `width`, a proportion of the width of the surrounding text). If you use character vectors, they must be valid CSS or LaTeX widths. The only unit both systems have in common is `pt` for points.

```
width(ht) <- 0.35
col_width(ht) <- c(.7, .3)
ht
```

Table 5: Employee table

Employee	Salary
John Smith	50000.00
Jane Doe	50000.00
Hadley Wickham	100000.00
David Hugh-Jones	40000.00

DHJ deserves a pay rise

It is best to set table width explicitly, then set column widths as proportions.

By default, if a cell contains long contents, it will be stretched. Use the `wrap` property to allow cell contents to wrap over multiple lines:

```
ht_wrapped <- ht
ht_wrapped[5, 1] <- 'David Arthur Shrimpton Hugh-Jones'
wrap(ht_wrapped) <- TRUE
ht_wrapped
```

Table 6: Employee table

Employee	Salary
John Smith	50000.00
Jane Doe	50000.00
Hadley Wickham	100000.00
David Arthur Shrimpton Hugh-Jones	40000.00

DHJ deserves a pay rise

Adding row and column names

Just like data frames, huxtables can have row and column names. Often, we want to add these to the final table. You can do this using either the `add_colnames`/`add_rownames` arguments to `as_huxtable`, or the `add_colnames()`/`add_rownames()` functions. (Note that earlier versions of `dplyr` used to have functions with the same name.)

```
as_hux(mtcars[1:4, 1:4])                              %>%
      huxtable::add_rownames(colname = 'Car name') %>%
      huxtable::add_colnames()
```

Car name	mpg	cyl	disp	hp
Mazda RX4	21	6	160	110
Mazda RX4 Wag	21	6	160	110
Datsun 710	22.8	4	108	93
Hornet 4 Drive	21.4	6	258	110

Column and row spans

Huxtable cells can span multiple rows or columns, using the `colspan` and `rowspan` properties.

```
cars_mpg <- cbind(car_type = rep("", nrow(cars_mpg)), cars_mpg)
cars_mpg$car_type[1] <- 'Four cylinders'
cars_mpg$car_type[13] <- 'Six cylinders'
cars_mpg$car_type[20] <- 'Eight cylinders'
rowspan(cars_mpg)[1, 1] <- 12
rowspan(cars_mpg)[13, 1] <- 7
rowspan(cars_mpg)[20, 1] <- 14

cars_mpg <- rbind(c('', 'List of cars', '', '', ''), cars_mpg)
colspan(cars_mpg)[1, 2] <- 4
align(cars_mpg)[1, 2] <- 'center'

# a little more formatting:

cars_mpg <- set_all_padding(cars_mpg, 2)
cars_mpg <- set_all_borders(cars_mpg, 1)
valign(cars_mpg)[1,] <- 'top'
col_width(cars_mpg) <- c(.4 , .3 , .1, .1, .1)
number_format(cars_mpg)[, 4:5] <- 0
bold(cars_mpg)[1:2, ] <- TRUE
bold(cars_mpg)[, 1] <- TRUE
if (is_latex) font_size(cars_mpg) <- 10
cars_mpg
```

	List of cars			
	Car	**mpg**	**cyl**	**am**
Four cylinders	Datsun 710	22.8	4	1
	Merc 240D	24.4	4	0
	Merc 230	22.8	4	0
	Fiat 128	32.4	4	1
	Honda Civic	30.4	4	1
	Toyota Corolla	33.9	4	1
	Toyota Corona	21.5	4	0
	Fiat X1-9	27.3	4	1
	Porsche 914-2	26	4	1
	Lotus Europa	30.4	4	1
	Volvo 142E	21.4	4	1
Six cylinders	Mazda RX4	21	6	1
	Mazda RX4 Wag	21	6	1
	Hornet 4 Drive	21.4	6	0
	Valiant	18.1	6	0
	Merc 280	19.2	6	0
	Merc 280C	17.8	6	0
	Ferrari Dino	19.7	6	1
Eight cylinders	Hornet Sportabout	18.7	8	0
	Duster 360	14.3	8	0
	Merc 450SE	16.4	8	0
	Merc 450SL	17.3	8	0
	Merc 450SLC	15.2	8	0
	Cadillac Fleetwood	10.4	8	0
	Lincoln Continental	10.4	8	0
	Chrysler Imperial	14.7	8	0
	Dodge Challenger	15.5	8	0
	AMC Javelin	15.2	8	0
	Camaro Z28	13.3	8	0
	Pontiac Firebird	19.2	8	0
	Ford Pantera L	15.8	8	1
	Maserati Bora	15	8	1

Quick themes

Huxtable comes with some predefined themes for formatting.

```
theme_striped(cars_mpg[14:20,], stripe = 'bisque1', header_col = FALSE, header_row = FALSE)
```

Six cylinders	Mazda RX4	21	6	1
	Mazda RX4 Wag	21	6	1
	Hornet 4 Drive	21.4	6	0
	Valiant	18.1	6	0
	Merc 280	19.2	6	0
	Merc 280C	17.8	6	0
	Ferrari Dino	19.7	6	1

Selecting rows, columns and cells

Row and column functions

If you use the `set_*` style functions, huxtable has some convenience functions for selecting rows and columns.

To select all rows, or all columns, use `everywhere` in the row or column specification. To select just even or odd-numbered rows or columns, use `evens` or `odds`. To select the last `n` rows or columns, use `final(n)`. To select every nth row, use `every(n)` and to do this starting from row m use `every(n, from = m)`.

With these functions it is easy to add striped backgrounds to tables:

```
car_ht                                                  %>%
      set_background_color(evens, everywhere, 'wheat') %>%
      set_background_color(odds, everywhere, grey(.9)) %>%
      set_bold(1, everywhere, TRUE)
```

Car	MPG	Cylinders	Horsepower	kml
Valiant	18.1	6	105	6.42
Mazda RX4	21	6	110	7.45
Mazda RX4 Wag	21	6	110	7.45
Hornet 4 Drive	21.4	6	110	7.59
Merc 280	19.2	6	123	6.81
Hornet Sportabout	18.7	8	175	6.63
Duster 360	14.3	8	245	5.07

Of course you could also just do `1:nrow(car_ht)`, but, in the middle of a dplyr pipe, you may not know exactly how many rows or columns you have. Also, these functions make your code easy to read.

You can also use `dplyr` functions like `starts_with()`, `contains()`, and `matches()` to specify columns by column name. For a full list of these functions, see `?select_helpers`.

```
car_ht %>% set_background_color(everywhere, starts_with('C'), 'orange')
```

Car	MPG	Cylinders	Horsepower	kml
Valiant	18.1	6	105	6.42
Mazda RX4	21	6	110	7.45
Mazda RX4 Wag	21	6	110	7.45
Hornet 4 Drive	21.4	6	110	7.59
Merc 280	19.2	6	123	6.81
Hornet Sportabout	18.7	8	175	6.63
Duster 360	14.3	8	245	5.07

```
car_ht %>% set_italic(everywhere, matches('[aeiou]'), TRUE)
```

Car	MPG	*Cylinders*	*Horsepower*	kml
Valiant	18.1	*6*	*105*	6.42
Mazda RX4	21	*6*	*110*	7.45
Mazda RX4 Wag	21	*6*	*110*	7.45
Hornet 4 Drive	21.4	*6*	*110*	7.59
Merc 280	19.2	*6*	*123*	6.81
Hornet Sportabout	18.7	*8*	*175*	6.63
Duster 360	14.3	*8*	*245*	5.07

Note that unlike in `dplyr`'s `select` function, you have to specify rows as well as columns.

Lastly, remember that you can set a property for every cell by simply omitting the `row` and `col` arguments, like this: `set_background_color(ht, 'orange')`.

Conditional formatting

You may want to apply conditional formatting to cells, based on their contents. Suppose we want to display a table of correlations, and to highlight ones which are significant. We can use the `where()` function to select those cells.

```
library(psych)
data(attitude)
att_corr <- corr.test(as.matrix(attitude))

att_hux <- as_hux(att_corr$r)                                              %>%
      # selects cells with p < 0.05:
      set_background_color(where(att_corr$p < 0.05), 'yellow')             %>%
      # selects cells with p < 0.01:
      set_background_color(where(att_corr$p < 0.01), 'orange')             %>%
      set_text_color(where(row(att_corr$r) == col(att_corr$r)), 'grey')


att_hux <- att_hux                                                         %>%
      huxtable::add_rownames()                                             %>%
      huxtable::add_colnames()                                             %>%
      set_caption('Correlations in attitudes among 30 departments')        %>%
      set_bold(1, everywhere, TRUE)                                        %>%
      set_bold(everywhere, 1, TRUE)                                        %>%
      set_all_borders(1)                                                   %>%
      set_number_format(2)                                                 %>%
      set_position('left')

att_hux
```

Table 7: Correlations in attitudes among 30 departments

rownames	rating	complaints	privileges	learning	raises	critical	advance
rating	1.00	0.83	0.43	0.62	0.59	0.16	0.16
complaints	0.83	1.00	0.56	0.60	0.67	0.19	0.22
privileges	0.43	0.56	1.00	0.49	0.45	0.15	0.34
learning	0.62	0.60	0.49	1.00	0.64	0.12	0.53
raises	0.59	0.67	0.45	0.64	1.00	0.38	0.57
critical	0.16	0.19	0.15	0.12	0.38	1.00	0.28
advance	0.16	0.22	0.34	0.53	0.57	0.28	1.00

We have now seen three ways to call `set_*` functions in huxtable:

- With four arguments, like `set_property(hux_object, rows, cols, value)`;
- With two arguments, like `set_property(hux_object, value)` to set a property everywhere;
- With three arguments, like `set_property(hux_object, where(condition), value)` to set a property for specific cells.

The second argument of the three-argument version must return a 2-column matrix. Each row of the matrix gives one cell. `where()` does this for you: it takes a logical matrix argument and returns the rows and columns where a condition is `TRUE`. It's easiest to show this with an example:

```
m <- matrix(c('dog', 'cat', 'dog', 'dog', 'cat', 'cat', 'cat', 'dog'), 4, 2)
m
```

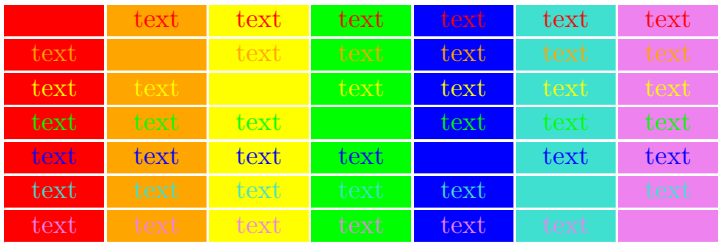
```
##      [,1]  [,2]
## [1,] "dog" "cat"
## [2,] "cat" "cat"
## [3,] "dog" "cat"
## [4,] "dog" "dog"
```

```
where(m == 'dog') # m is equal to 'dog' in cells (1, 1), (3, 1), (4, 1) and (4, 2):
```

```
##      row col
## [1,]   1   1
## [2,]   3   1
## [3,]   4   1
## [4,]   4   2
```

`set_*` functions have one more optional argument, the `byrow` argument, which is `FALSE` by default. If you set a single pattern for many cells, you may want the pattern to fill the matrix by column or by row. The default fills the pattern in going down columns. If you set `byrow = TRUE`, the pattern goes across rows instead. (This is a bit confusing: typically, `byrow = TRUE` means that the *columns* will all look the same. But it works the same way as the `byrow` argument to `matrix()`.)

```
color_demo <- matrix('text', 7, 7)
rainbow <- c('red', 'orange', 'yellow', 'green', 'blue', 'turquoise', 'violet')
color_demo <- as_hux(color_demo)                          %>%
      set_text_color(rainbow)                             %>% # text rainbow down columns
      set_background_color(rainbow, byrow = TRUE) %>% # background color rainbow along rows
      set_all_borders(1)                                  %>%
      set_all_border_colors('white')
color_demo
```



Creating a regression table

A common task for scientists is to create a table of regressions. The function `huxreg` does this for you. Here's a quick example:

```
data(diamonds, package = 'ggplot2')

lm1 <- lm(price ~ carat, diamonds)
lm2 <- lm(price ~ depth, diamonds)
lm3 <- lm(price ~ carat + depth, diamonds)

huxreg(lm1, lm2, lm3)
```

	(1)	(2)	(3)
(Intercept)	-2256.361 ***	5763.668 ***	4045.333 ***
	(13.055)	(740.556)	(286.205)
carat	7756.426 ***		7765.141 ***
	(14.067)		(14.009)
depth		-29.650 *	-102.165 ***
		(11.990)	(4.635)
N	53940	53940	53940
R2	0.849	0.000	0.851
logLik	-472730.266	-523772.431	-472488.441
AIC	945466.532	1047550.862	944984.882

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

For more information see the `huxreg` vignette, available online in HTML or PDF or in R via `vignette('huxreg')`.

Output to different formats

Automatic pretty-printing of data frames

If you load huxtable within a knitr document, it will automatically format data frames for you by installing a `knit_print.data_frame` command.

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4        21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag    21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710       22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive   21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

```
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

If you don't want this (e.g. if you want to use `knitr::kable` or the printr package, then you can turn it off like this:

```
options(huxtable.knit_print_df = FALSE)

head(mtcars) # back to normal
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Using huxtables in knitr and rmarkdown

If you use knitr and rmarkdown in RStudio, huxtable objects should automatically display in the appropriate format (HTML or LaTeX). You need to have some LaTeX packages installed for huxtable to work. To find out what these are, you can call `report_latex_dependencies()`. This will print out and/or return 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.

Rmarkdown exports to Word via Markdown. You can use huxtable to do this, but since Markdown tables are rather basic, a lot of formatting will be lost. If you want to create Word or Powerpoint documents directly, install the flextable package from CRAN. You can then convert your huxtable objects to `flextable` objects and include them in Word or Powerpoint documents. Almost all formatting should work. See the `flextable` and `officer` documentation and `?as_flextable` for more details.

Similarly, to create formatted reports in Excel, install the openxlsx package. You can then use `as_Workbook` to convert your huxtables to Workbook objects, and save them using `openxlsx::saveWorkbook`.

Sometimes you may want to select how huxtable objects are printed by default. For example, in an RStudio notebook (a .Rmd document with `output_format = html_notebook`), huxtable can't automatically work out what format to use, as of the time of writing. You can set it manually using `options(huxtable.print = print_notebook)` which prints out HTML in an appropriate format.

You can print a huxtable on screen using `print_screen` (or just 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(ht)
```

```
##          Employee table
##   Employee              Salary
## ──────────────────────────────
##   John Smith          50000.00
##   Jane Doe            50000.00
##   Hadley Wickham     100000.00
##   David Hugh-Jones    40000.00
## ──────────────────────────────
##   DHJ deserves a pay rise
##
## Column names: Employee, Salary
```

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 Word, HTML or PDF document. To do this you can use the `quick_docx`, `quick_html`, `quick_pdf` and `quick_xlsx` functions. 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. By default the file will be in the current directory under the name e.g. `huxtable-output.pdf`. If the file already exists, you'll be asked to confirm the overwrite. For non-interactive use, you must specify a filename yourself explicitly – this keeps you from accidentally trashing your files.

```
quick_pdf(mtcars)
quick_pdf(mtcars, file = 'motorcars data.pdf')
```

End matter

For more information, see the website or github.