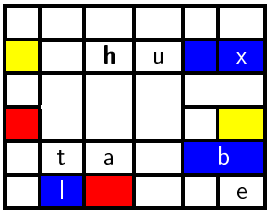
Introduction to Huxtable

David Hugh-Jones

2019-03-20

Contents



Introduction

About this document

This is the introductory vignette for the R package 'huxtable', version 4.4.0.9000. 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. 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 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 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, and `mydata$start_time` is all the data in the `start_time` column.

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. This works very much like `data.frame`:

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

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 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 the data will start on row 2 of the huxtable, and the column names will be row 1.

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

```
##   Employee            Salary
##   John Smith           50000
##   Jane Doe             50000
##   David Hugh-Jones     40000
##
## Column names: Employee, Salary
```

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:

```
ht
```

Employee	Salary
John Smith	50000
Jane Doe	50000
David Hugh-Jones	40000

Changing the look and feel

Huxtable properties

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

To do this, we will set some **properties** on the table cells. 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
```

We can set the `number_format` property to change how numbers are displayed:

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

To assign properties to just some cells, you use subsetting, 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_height`, and one column property, `col_width`.

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			caption_pos
font	bottom_border			height
font	bottom_border_color			label
font_size	bottom_border_style			latex_float
italic	bottom_padding			position
na_string	colspan			tabular_environment
number_format	left_border			width
rotation	left_border_color			
text_color	left_border_style			
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			

Tidyverse syntax

If you prefer a tidyverse style of code, using the pipe operator `%>%`, then you can use `set_*` functions to set properties. These are named like `set_xxx` where `xxx` is the property name, for example `set_bold` for the `bold` property.

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

Table 3: Employee table

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

`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)`.

There are also some 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_border_styles` sets left, right, top and bottom border styles;
- `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"
```

Editing content

Changing and adding content

You can treat a huxtable just like a data frame. If you want to change data in a cell, assign to that cell:

```
ht[3, 1] <- 'Jane Jones'
ht
```

Table 4: Employee table

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

To add a column, do, e.g.:

```
ht_with_roles <- ht
ht_with_roles$Role <- c("Role", "Admin", "CEO", "Dogsbody")
ht_with_roles
```

Table 5: Employee table

Employee	**Salary**	Role
John Smith	50000.00	Admin
Jane Jones	50000.00	CEO
David Hugh-Jones	40000.00	Dogsbody

Notice that the third column doesn't have the properties we set on the first two columns, like the bold first row and the underlining.

If we want new columns to copy properties from their neighbours, we can use `cbind`, a base R function that binds columns together. When you `cbind` huxtable objects, by default, cell properties are copied over from their neighbours:

```
ht_with_roles <- cbind(ht, c("Role", "Admin", "CEO", "Dogsbody"))
ht_with_roles
```

Table 6: Employee table

Employee	**Salary**	**Role**
John Smith	50000.00	Admin
Jane Jones	50000.00	CEO
David Hugh-Jones	40000.00	Dogsbody

`rbind` works the same way:

```
rbind(ht, c("Yihui Xie", 100000))
```

Table 7: Employee table

Employee	**Salary**
John Smith	50000.00
Jane Jones	50000.00
David Hugh-Jones	40000.00
Yihui Xie	100000.00

Notice how Yihui's salary has got the same number formatting as the other employees. That's because cell properties for the new row were copied from the row above.

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:

```
to_insert <- hux(
        Role  = c("Admin", "CEO", "Dogsbody"),
        Hired = as.Date(c("2015-01-01", "2008-06-05", "2012-07-31")),
        add_colnames = TRUE
      ) %>%
      set_bold(1, 1:2, TRUE) %>%
      set_bottom_border(1, 1:2, TRUE)

cbind(ht[, 1], to_insert, ht[, 2:ncol(ht)])
```

Table 8: Employee table

Employee	Role	Hired	Salary
John Smith	Admin	2015-01-01	50000.00
Jane Jones	CEO	2008-06-05	50000.00
David Hugh-Jones	Dogsbody	2012-07-31	40000.00

It's more elegant to use huxtable's `add_columns` function:

```
add_columns(ht, to_insert, after = 1)
```

Table 9: Employee table

Employee	Role	Hired	Salary
John Smith	Admin	2015-01-01	50000.00
Jane Jones	CEO	2008-06-05	50000.00
David Hugh-Jones	Dogsbody	2012-07-31	40000.00

The `after` argument says where the second object should be inserted. `after` can also be a column name:

```
add_columns(ht, to_insert, after = "Employee")
```

Table 10: Employee table

Employee	Role	Hired	Salary
John Smith	Admin	2015-01-01	50000.00
Jane Jones	CEO	2008-06-05	50000.00
David Hugh-Jones	Dogsbody	2012-07-31	40000.00

The function `add_rows` works in the same way.

Editing content the base R way

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

```
car_ht <- as_hux(mtcars)
car_ht <- huxtable::add_rownames(car_ht, "Car")
# Select columns by name:
car_ht <- car_ht[, c("Car", "mpg", "cyl", "am")]

# Order by number of cylinders:
car_ht <- car_ht[order(car_ht$cyl), ]

car_ht <- huxtable::add_colnames(car_ht)

# Pretty output, see below:
theme_plain(car_ht[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

Editing content the dplyr way

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

```
car_ht <- as_hux(mtcars)
car_ht <- huxtable::add_rownames(car_ht, colname = "Car")

car_ht <- car_ht                                              %>%
      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)                                  %>%
      huxtable::add_colnames()

theme_plain(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.

More formatting

Number format

You can change how huxtable formats numbers using `number_format`. Set `number_format` to a number of decimal places. This affects any numbers found within the cell text.

```
htn <- hux(c(
        "Some numbers...",
        11.003,
        300,
        12.02,
        "12.1 **",
        "mean 11.7 (se 2.3)"
```

```
      ))

number_format(htn) <- 3
theme_plain(htn)
```

Some numbers...
11.003
300.000
12.020
12.100 **
mean 11.700 (se 2.300)

To align columns by decimal places, just set the `align` property to `"."`:

```
align(htn)[2:6, ] <- "." # not the first row

theme_plain(htn)
```

Some numbers...
11.003
300.000
12.020
12.100 **
mean 11.700 (se 2.300)

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 = c(8.9, 9.2, 7.8)
      )

as_hux(my_data, add_colnames = TRUE) # with automatic formatting
```

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

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

Employee	Salary	Performance
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;"))

theme_plain(code_ht)
```

Some maths
a^b

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

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

theme_plain(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) <- 0.4
col_width(ht) <- c(.7, .3)
ht
```

Table 11: Employee table

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

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 12: Employee table

Employee	Salary
John Smith	50000.00
Jane Jones	50000.00
David Hugh-Jones	40000.00
David Arthur Shrimpton Hugh-Jones	

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.

```
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

Merging cells

Sometimes you want a single cell to spread over more than one row or column: for example, if you want a heading that covers several different rows.

You can do this by calling `merge_cells(ht, rows, cols)`. `rows` and `cols` should be a contiguous sequence of numbers. The rectangle of cells `ht[rows, cols]` will be merged.

When cells in a rectangle are merged, all cells apart from the top left one are hidden, along with any properties they have. So if you want to set cell properties, you have to target the top left cell.

Here, we'll add some row and column headings to the `mtcars` dataset:

```
car_ht <- as_hux(mtcars)
car_ht <- huxtable::add_rownames(car_ht, colname = "Car")
car_ht <- car_ht %>% arrange(cyl) %>% select(1:4)
car_ht <- huxtable::add_colnames(car_ht)

car_ht <- cbind(cylinders = car_ht$cyl, car_ht)
car_ht$cylinders[1]   <- ""
car_ht$cylinders[2]   <- "Four cylinders"
car_ht$cylinders[13]  <- "Six cylinders"
car_ht$cylinders[20]  <- "Eight cylinders"

car_ht <- car_ht %>%
  merge_cells(2:12, 1) %>%
  merge_cells(13:19, 1) %>%
  merge_cells(20:33, 1)


car_ht <- rbind(c("List of cars", "", "", "", ""), car_ht)
car_ht <- merge_cells(car_ht, 1, 1:5)
align(car_ht)[1, 1] <- "center"

car_ht <- theme_plain(car_ht)
right_border(car_ht)[1, 1] <- 0.4
bottom_border(car_ht)[21, 1] <- 0.4
car_ht
```

List of cars				
	Car	mpg	cyl	disp
Four cylinders	Datsun 710	22.8	4	108
	Merc 240D	24.4	4	147
	Merc 230	22.8	4	141
	Fiat 128	32.4	4	78.7
	Honda Civic	30.4	4	75.7
	Toyota Corolla	33.9	4	71.1
	Toyota Corona	21.5	4	120
	Fiat X1-9	27.3	4	79
	Porsche 914-2	26	4	120
	Lotus Europa	30.4	4	95.1
	Volvo 142E	21.4	4	121
Six cylinders	Mazda RX4	21	6	160
	Mazda RX4 Wag	21	6	160
	Hornet 4 Drive	21.4	6	258
	Valiant	18.1	6	225
	Merc 280	19.2	6	168
	Merc 280C	17.8	6	168
	Ferrari Dino	19.7	6	145
Eight cylinders	Hornet Sportabout	18.7	8	360
	Duster 360	14.3	8	360
	Merc 450SE	16.4	8	276
	Merc 450SL	17.3	8	276
	Merc 450SLC	15.2	8	276
	Cadillac Fleetwood	10.4	8	472
	Lincoln Continental	10.4	8	460
	Chrysler Imperial	14.7	8	440
	Dodge Challenger	15.5	8	318
	AMC Javelin	15.2	8	304
	Camaro Z28	13.3	8	350
	Pontiac Firebird	19.2	8	400
	Ford Pantera L	15.8	8	351
	Maserati Bora	15	8	301

Quick themes

Huxtable comes with some predefined themes for formatting. You've already seen `theme_plain`. Other options include `theme_basic` and (for fun) `theme_mondrian`:

```
theme_mondrian(ht)
```

Table 13: Employee table

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

The "themes" vignette shows all the available themes.

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 the last `n` rows or columns, use `final(n)`.

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`.

```
ht %>% set_background_color(everywhere, starts_with("S"), "orange")
```

Table 14: Employee table

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

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

Conditional formatting

When you want to apply different formatting to different cells, you can use the `map_xxx` functions.

For example, we could create a striped table like this:

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

Table 15: Employee table

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

Or, we could apply a text color to our cars based on their ecological performance:

```
car_ht %>% map_text_color(everywhere, 3,
        by_ranges(c(15, 25), c("red", "darkgreen", "green"))
      )
```

List of cars				
	Car	mpg	cyl	disp
Four cylinders	Datsun 710	22.8	4	108
	Merc 240D	24.4	4	147
	Merc 230	22.8	4	141
	Fiat 128	32.4	4	78.7
	Honda Civic	30.4	4	75.7
	Toyota Corolla	33.9	4	71.1
	Toyota Corona	21.5	4	120
	Fiat X1-9	27.3	4	79
	Porsche 914-2	26	4	120
	Lotus Europa	30.4	4	95.1
	Volvo 142E	21.4	4	121
Six cylinders	Mazda RX4	21	6	160
	Mazda RX4 Wag	21	6	160
	Hornet 4 Drive	21.4	6	258
	Valiant	18.1	6	225
	Merc 280	19.2	6	168
	Merc 280C	17.8	6	168
	Ferrari Dino	19.7	6	145
Eight cylinders	Hornet Sportabout	18.7	8	360
	Duster 360	14.3	8	360
	Merc 450SE	16.4	8	276
	Merc 450SL	17.3	8	276
	Merc 450SLC	15.2	8	276
	Cadillac Fleetwood	10.4	8	472
	Lincoln Continental	10.4	8	460
	Chrysler Imperial	14.7	8	440
	Dodge Challenger	15.5	8	318
	AMC Javelin	15.2	8	304
	Camaro Z28	13.3	8	350
	Pontiac Firebird	19.2	8	400
	Ford Pantera L	15.8	8	351
	Maserati Bora	15	8	301

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

- `by_rows` applies different properties to different rows in sequence.
- `by_ranges` divides its input into ranges, and outputs colours based on that, like the base R function `cut`.

Above, we mapped all numbers below 15 to `"red"`, numbers between 15 and 25 to `"darkgreen"` and numbers above 25 to `"green"`.

The syntax of all `map_xxx` functions is `map_xxx(ht, row, col, fn)`. `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`.

You can also apply properties for cells that match a string, using the `by_regex` mapping function:

```
ht %>% map_bold(by_regex('Jones' = TRUE))
```

Table 16: Employee table

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

There is a `map_xxx` function for each property `xxx`. See the help pages for other mapping functions.

Creating a regression table

To create a table of regressions, use `huxreg`:

```
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.

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:

```
head(mtcars)
```

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21	6	160	110	3.9	2.62	16.5	0	1	4	4
21	6	160	110	3.9	2.88	17	0	1	4	4
22.8	4	108	93	3.85	2.32	18.6	1	1	4	1
21.4	6	258	110	3.08	3.21	19.4	1	0	3	1
18.7	8	360	175	3.15	3.44	17	0	0	3	2
18.1	6	225	105	2.76	3.46	20.2	1	0	3	1

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(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, 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.

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. 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(ht)
```

```
##          Employee table
##   Employee              Salary
## ──────────────────────────────
##   John Smith          50000.00
##   Jane Jones          50000.00
##   David Hugh-Jones    40000.00
##
## 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 document. To do this you can use huxtable functions starting with `quick_`:

Command	Output
`quick_pdf`	PDF document
`quick_docx`	Word document
`quick_html`	HTML web page
`quick_xlsx`	Excel spreadsheet
`quick_pptx`	Powerpoint presentation
`quick_rtf`	RTF document

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 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.