

# mmtable2

## A ggplot2-syntax for making tables



(Click image to play tutorial)

## mmtable2

### This R package uses **ggplot2** syntax to create great tables

I love `ggplot2` for plotting. The grammar of graphics allows us to add elements to plots. Tables seem to be forgotten in terms of an intuitive grammar with tidy data philosophy – Until now. `mmtable2` aims to be the `ggplot2` for tables, leveraging the awesome `GT` table package.

The `mmtable2` package aims to make it easy to create tables by:

1. Using a `ggplot2`-style syntax for using a grammar of table operations.
2. Extends the amazing `GT` table package.

Here's what we're making today:

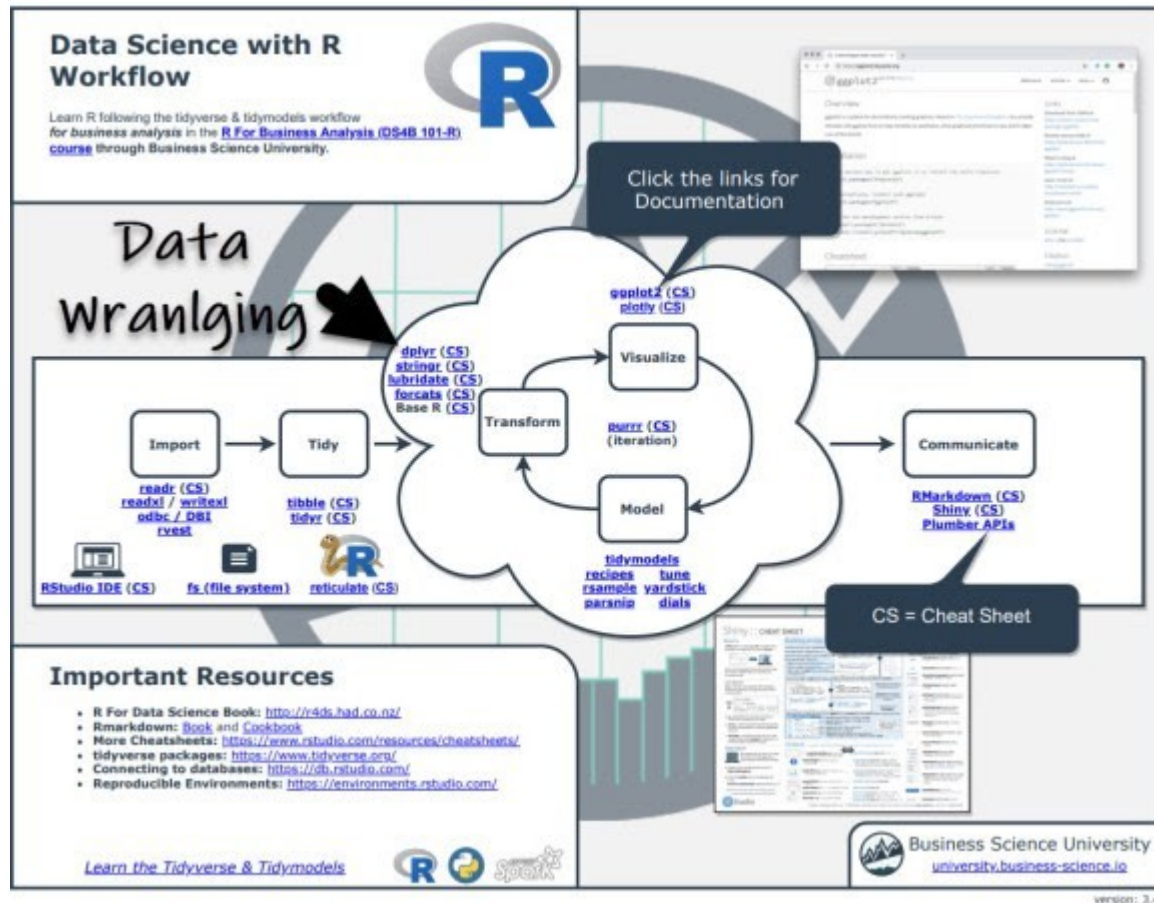
|     |   | Audi | Chevrolet | Dodge | Ford | Honda | Hyundai | Jeep | Land Rover | Lincoln | Mercury | Nissan | Pontiac | Subaru | Toyota | Volkswagen |
|-----|---|------|-----------|-------|------|-------|---------|------|------------|---------|---------|--------|---------|--------|--------|------------|
| CTY | 4 | 19.1 | 20.5      | 18    |      | 24.4  | 19.5    |      |            |         |         | 21.5   |         | 19.3   | 20.9   | 22.5       |
|     | 5 |      |           |       |      |       |         |      |            |         |         |        |         |        |        | 20.5       |
|     | 6 | 16.4 | 17.7      | 15    | 15.3 |       | 17.5    | 15.7 |            |         | 13.5    | 17.1   | 17.2    |        | 16.6   | 16.8       |
|     | 8 | 16   | 13.6      | 11.6  | 13.1 |       |         | 12.2 | 11.5       | 11.3    | 13      | 12     | 16      |        | 12.7   |            |
| HWY | 4 | 28.1 | 28.5      | 24    |      | 32.6  | 28      |      |            |         |         | 29.8   |         | 25.6   | 28.2   | 30.9       |
|     | 5 |      |           |       |      |       |         |      |            |         |         |        |         |        |        | 28.8       |
|     | 6 | 25.3 | 27        | 20.7  | 20.7 |       | 25.3    | 20.3 |            |         | 18      | 22.9   | 26.8    |        | 22.2   | 24.8       |
|     | 8 | 23   | 19.9      | 15.7  | 18.5 |       |         | 16   | 16.5       | 17      | 18      | 18     | 25      |        | 16.7   |            |

Make professional tables using a `ggplot`-syntax

# Before we get started, get the Cheat Sheet


`mmtable2` is great for making tables fast. But, you'll still need to learn how to wrangle data. For those topics, I'll use the [Ultimate R Cheat Sheet](#) to refer to `dplyr` code in my workflow.

**Quick example – Clicking the “CS” next to “dplyr” opens the Data Transformation with Dplyr Cheat Sheet.**



Now you're ready to quickly reference `dplyr` functions. Ok, onto the tutorial.

# Data Transformation with dplyr : : CHEAT SHEET



**dplyr** functions work with pipes and expect **tidy data**. In tidy data:

- Each **variable** is in its own **column**
- Each **observation**, or **case**, is in its own **row**
- $x \%>\% f(y)$  becomes  $f(x, y)$

**Summarise Cases**

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

**summary function**

**summarise(data, ...)**  
Compute table of summaries.  
`summarise(mtcars, avg = mean(mpg))`

**count(x, ..., wt = NULL, sort = FALSE)**  
Count number of rows in each group defined by the variables in .... Also **tally()**.  
`count(mtcars, Species)`

**VARIATIONS**

- summarise\_all()** - Apply funs to every column.
- summarise\_at()** - Apply funs to specific columns.
- summarise\_if()** - Apply funs to all cols of one type.

**Group Cases**

Use **group\_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

`mtcars %>%  
group_by(cyl) %>%  
summarise(avg = mean(mpg))`

**group\_by(data, ..., add = FALSE)**  
Returns copy of table grouped by ...  
`g_mtcars <- group_by(mtcars, Species)`

**ungroup(x, ...)**  
Returns ungrouped copy of table.  
`ungroup(g_mtcars)`

**Logical and boolean operators to use with filter()**

|   |    |         |      |   |       |
|---|----|---------|------|---|-------|
| < | <= | is.na() | %in% |   | xor() |
| > | >= | is.na() |      | & |       |

See **Rbase::Logic** and **TComparison** for help.

**Manipulate Cases**

**EXTRACT CASES**  
Row functions return a subset of rows as a new table.

- filter(data, ...)** Extract rows that meet logical criteria. `filter(mtcars, Sepal.Length > 7)`
- distinct(data, ..., keep\_all = FALSE)** Remove rows with duplicate values. `distinct(mtcars, Species)`
- sample\_frac(n, size = 1, replace = FALSE, weight = NULL, env = parent.frame())** Randomly select fraction of rows. `sample_frac(mtcars, 0.5, replace = TRUE)`
- sample\_n(n, size, replace = FALSE, weight = NULL, env = parent.frame())** Randomly select size rows. `sample_n(mtcars, 20, replace = TRUE)`
- slice(data, ...)** Select rows by position. `slice(mtcars, 20:25)`
- top\_n(x, n, wt)** Select and order top n entries (by group if grouped data). `top_n(mtcars, 5, Sepal.Length)`

**Manipulate Variables**

**EXTRACT VARIABLES**  
Column functions return a set of columns as a new vector or table.

- pull(data, var = -1)** Extract column values as a vector. Choose by name or index. `pull(mtcars, Sepal.Length)`
- select(data, ...)** Extract columns as a table. Also **select\_if()**. `select(mtcars, Sepal.Length, Species)`

**Use these helpers with select()**  
e.g. `select(mtcars, starts_with("Sepal"))`

- contains(match)** **num\_range(prefix, range)** e.g. `mpg:cyl`
- ends\_with(match)** **one\_of(...)** e.g. `Species`
- matches(match)** **starts\_with(match)**

**MAKE NEW VARIABLES**

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

**vectorized function**

- mutate(data, ...)** Compute new column(s). `mutate(mtcars, gpm = 1/mpg)`
- transmute(data, ...)** Compute new column(s), drop others. `transmute(mtcars, gpm = 1/mpg)`
- mutate\_all(tbl, funs, ...)** Apply funs to every column. Use with **funs()**. Also **mutate\_if()**. `mutate_all(mtcars, funs(log10, log2))`
- mutate\_at(tbl, cols, funs, ...)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for **select()**. `mutate_at(mtcars, vars(Species), funs(log10))`
- add\_column(data, ..., before = NULL, after = NULL)** Add new column(s). Also **add\_count()**, **add\_tally()**. `add_column(mtcars, new = 2:32)`
- rename(data, ...)** Rename columns. `rename(mtcars, Length = Sepal.Length)`

**R Studio**

RStudio® is a trademark of RStudio, Inc. • CC BY-SA RStudio • info@rstudio.com • 844-449-1212 • rstudio.com • Learn more with browsing.rstudio.com/package = c("dplyr", "tidyr") • dplyr 0.7.0 • tidyr 1.0.0 • Updated 2020-08

## Step 1: Load Libraries

The libraries we'll need today are mmttable2, gt, and tidyverse. As of this post, mmttable2 is not on CRAN so you'll need to install with github.

```
6 # LIBRARIES ----
7
8 # remotes::install_github("ianmoran11/mmttable2")
9
10 library(mmttable2)
11 library(gt)
12 library(tidyverse)
13
```

## Step 2: Wrangle Data into Long Format

Like ggplot2, mmttable2 standardizes on the long-format (tidy-data format). According to the tidyr vignette:

In **tidy data**:

1. Every column is a variable.
2. Every row is an observation.
3. Every cell is a single value.

To achieve the "tidy-data" format we need to leverage dplyr and tidyr (use the [Ultimate R Cheatsheet](#) to pull up data wrangling doc's).

### A. We start with Raw Data

This is the mpg data set, which contains fuel economy and other attributes on a number of automobile manufacturers and car models.

```
> mpg
# A tibble: 234 x 11
  manufacturer model      displ  year   cyl trans      drv      cty   hwy fl      class
  <chr>         <chr>    <dbl> <int> <int> <chr>    <chr> <int> <int> <chr> <chr>
1 audi         a4          1.8  1999     4 auto(l5) f        18    29 p    compact
2 audi         a4          1.8  1999     4 manual(m5) f        21    29 p    compact
3 audi         a4          2    2008     4 manual(m6) f        20    31 p    compact
4 audi         a4          2    2008     4 auto(av) f        21    30 p    compact
5 audi         a4          2.8  1999     6 auto(l5) f        16    26 p    compact
6 audi         a4          2.8  1999     6 manual(m5) f        18    26 p    compact
7 audi         a4          3.1  2008     6 auto(av) f        18    27 p    compact
8 audi         a4 quattro  1.8  1999     4 manual(m5) 4        18    26 p    compact
9 audi         a4 quattro  1.8  1999     4 auto(l5) 4        16    25 p    compact
10 audi        a4 quattro  2    2008     4 manual(m6) 4        20    28 p    compact
# ... with 224 more rows
```

## B. We tidy with dplyr and tidyr

This is a standard data wrangling operation. I teach data wrangling in-depth in the [R for Business Analysis course](#).

```
17 data_wrangled <- mpg %>%
18   group_by(manufacturer, cyl) %>%
19   summarise(across(.cols = c(cty, hwy), .fns = mean)) %>%
20   ungroup() %>%
21   pivot_longer(
22     cols = c(cty, hwy),
23     names_to = "fuel_economy_type",
24     values_to = "fuel_economy"
25   )
```

## C. And we output “tidy data”

The data is now in “tidy” format, ready for making a table. Every column is a variable, every row is an observation, every cell is a single value.

```
> data_wrangled
# A tibble: 64 x 4
  manufacturer      cyl fuel_economy_type fuel_economy
  <chr>          <int> <chr>                <dbl>
1 audi           4 cty                19.1
2 audi           4 hwy                28.1
3 audi           6 cty                16.4
4 audi           6 hwy                25.3
5 audi           8 cty                 16
6 audi           8 hwy                 23
7 chevrolet      4 cty                20.5
8 chevrolet      4 hwy                28.5
9 chevrolet      6 cty                17.7
10 chevrolet     6 hwy                 27
# ... with 54 more rows
```

[Get the Code](#)



## Step 3: Make the table with mmtable2

The data is now wrangled into the tidy format. We can use `mmtable2` to make the plot. To see `mmtable2` in action, I have a [full-tutorial on YouTube](#). The important points are:

- **mmtable()** – The main argument (other than the incoming data) is our value column. In our case it's `fuel_economy`, the measure of average vehicle fuel efficiency.
- **Header Top and Header Top Left** – These add column headers from features.
- **Head Left and Header Left Top** – These add row headers from features.
- **Header Format and Table Format** – These allow you to apply GT formatting functions.

```
30 # 2.0 Table Main ----
31 main_table <- data_wrangled %>%
32   mutate(fuel_economy = round(fuel_economy, 1)) %>%
33
34   mmtable(table_data = fuel_economy, table_name = "Fuel Economy") +
35
36   # Specify Headers
37   header_top(manufacturer) +
38   header_left(cyl) +
39   header_left_top(fuel_economy_type) +
40
41
42   # Specify formatting
43   header_format(manufacturer, list(cell_text(transform = "capitalize"))) +
44   header_format(fuel_economy_type, list(cell_text(transform = "uppercase")))
45   table_format(
46     locations = list(
47       cells_body(rows = c(2, 6))
48     )
49   )
```

[Get the Code](#)

And here is the professional table that was created, perfect for reports.

| Fuel Economy by Car Manufacturer                |   |      |           |       |      |       |         |      |            |         |         |        |         |        |        |            |
|---|---|------|-----------|-------|------|-------|---------|------|------------|---------|---------|--------|---------|--------|--------|------------|
| Audi, VW, and Honda are leaders in Fuel Economy |   |      |           |       |      |       |         |      |            |         |         |        |         |        |        |            |
|   |   | Audi | Chevrolet | Dodge | Ford | Honda | Hyundai | Jeep | Land Rover | Lincoln | Mercury | Nissan | Pontiac | Subaru | Toyota | Volkswagen |
| CTY   | 4 | 19.1 | 20.5      | 18    |      | 24.4  | 19.5    |      |            |         |         | 21.5   |         | 19.3   | 20.9   | 22.5       |
|   | 5 |      |           |       |      |       |         |      |            |         |         |        |         |        |        | 20.5       |
|   | 6 | 16.4 | 17.7      | 15    | 15.3 |       | 17.5    | 15.7 |            |         | 13.5    | 17.1   | 17.2    |        | 16.6   | 16.8       |
| HWY   | 8 | 16   | 13.6      | 11.6  | 13.1 |       |         | 12.2 | 11.5       | 11.3    | 13      | 12     | 16      |        | 12.7   |            |
|   | 4 | 28.1 | 28.5      | 24    |      | 32.6  | 28      |      |            |         |         | 29.8   |         | 25.6   | 28.2   | 30.9       |
|   | 5 |      |           |       |      |       |         |      |            |         |         |        |         |        |        | 28.8       |
|   | 6 | 25.3 | 27        | 20.7  | 20.7 |       | 25.3    | 20.3 |            |         | 18      | 22.9   | 26.8    |        | 22.2   | 24.8       |
|   | 8 | 23   | 19.9      | 15.7  | 18.5 |       |         | 16   | 16.5       | 17      | 18      | 18     | 25      |        | 16.7   |            |

Code available in our [Free R-Tips Github Repository](#)

## In Summary

You just quickly made a professional table using the ggplot2-style table package, **mmtable2**. This is an amazing accomplishment!!

You should be proud.