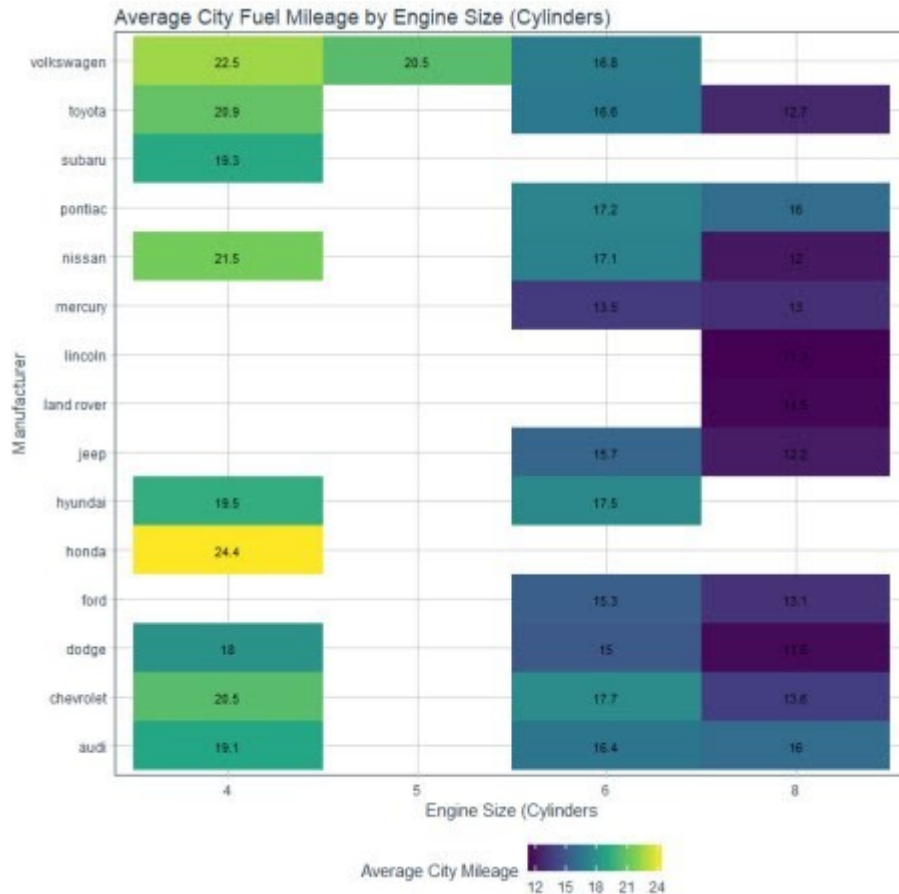


The data.table backend to dplyr

There's a new R package in town. It's called `dtplyr`. It's the `data.table` backend to `dplyr`. And, what it gets you is truly amazing:

- Enjoy the 3X to 5X `data.table` speedup with grouped summarizations
- All from the comfort of `dplyr`

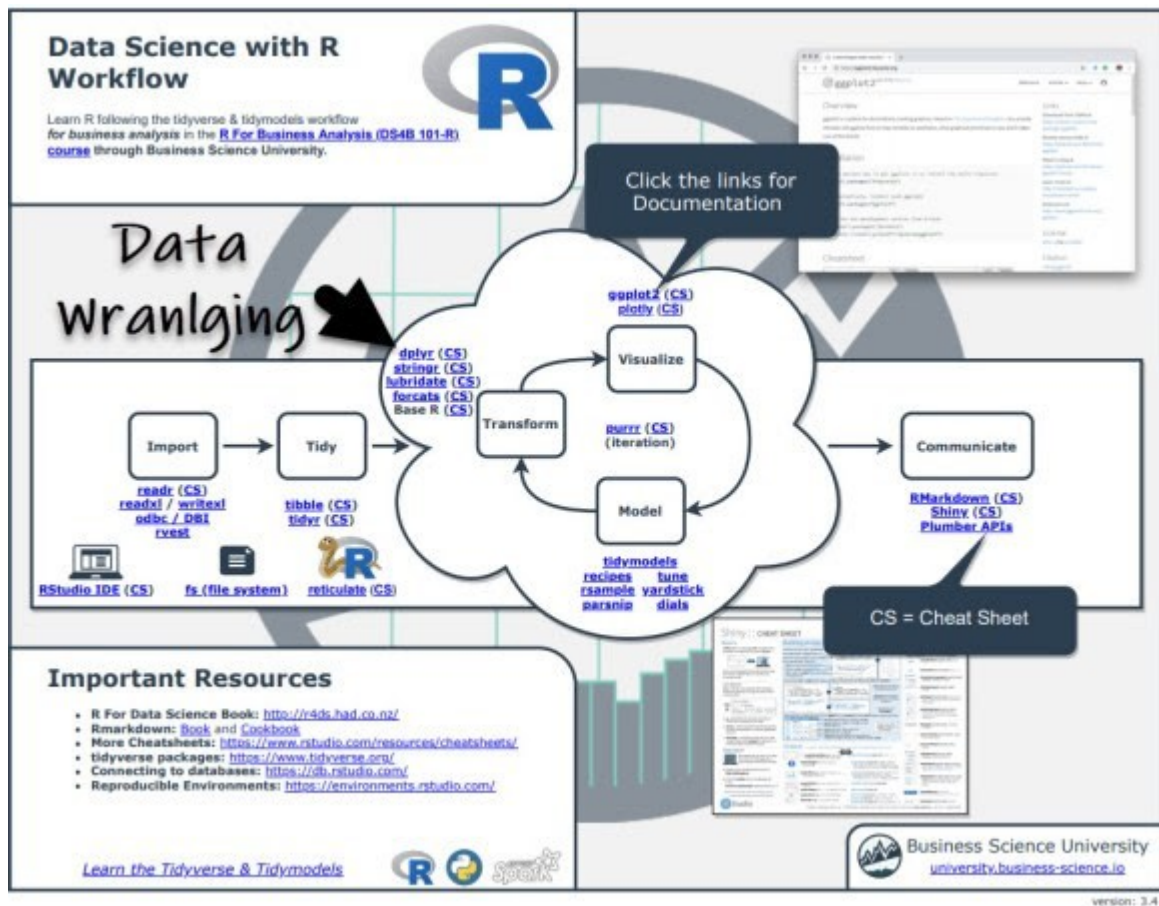


Make insanely fast grouped summaries by leveraging `data.table` with `dtplyr` then quickly visualize your summaries with `ggplot2`.

Before we get started, get the Cheat Sheet

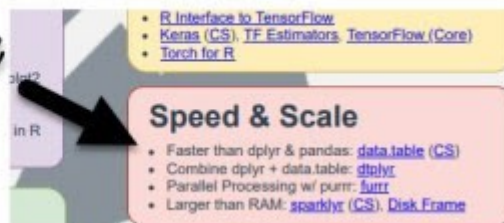
The most powerful tool in my arsenal is NOT my knowledge of the key R packages, but it's **knowing where to find R packages and documentation.**

The [Ultimate R Cheat Sheet](#) consolidates the documentation on every package I use frequently (including `dplyr`, `data.table`, and `dtplyr`).



If you tab through to page 3, you'll see a section called “Speed and Scale”. You can quickly see options to help including `data.table`, `dtplyr`, `furrr`, `sparklyr`, and `disk.frame`. Enjoy.

Data Table & Dtplyr



Get started with dtplyr

The first thing you'll want to do is **set up a Lazy Data Table** using the `lazy_dt()` function.

```

12 # 1.0 DATA TABLE ----
13
14 # * Make a Lazy Data Table ----
15 mpg_dt <- lazy_dt(mpg)
16 mpg_dt
17

```

So what happened? We now have a pointer to a `data.table`. This is a special connection that we can use to write `dplyr` code that gets converted to `data.table` code.

Lazy Data Table

```
> mpg_dt
Source: local data table [234 x 11]
Call: `DT4`
```

Sets up local
Data Table backend

	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(15)	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(15)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact

... with 228 more rows

Use as.data.table()/as.data.frame()/as_tibble() to access results

Translating dplyr to data.table

This idea of a data.table backend to dplyr is insanely powerful. Here's an example of a dplyr grouped-summarization that gets translated to data.table for a speedup.

- Start with lazy datatable connection object
- **Group by** the manufacturer and cylinder columns
- **Summarize** with the new `dplyr::across()` function
- **Ungroup** the lazy data.table

Dplyr Code

```
17 # * Summarize with Across ----
18 mpg_summary_dt <- mpg_dt %>%
19   group_by(manufacturer, cyl) %>%
20   summarise(across(
21     .cols = c(displ, cty:hwy),
22     .fns = list(mean, median),
23     .names = "{.fn}_{.col}"
24   )) %>%
25   ungroup()
26
27 mpg_summary_dt
```

The dtplyr backend does the heavy-lifting, converting your dplyr code into data.table code.

Data.Table Translation

```
> mpg_summary_dt %>% show_query()
`DT4`[, .(mean_displ = mean(displ), median_displ = median(displ),
  mean_cty = mean(cty), median_cty = median(cty), mean_hwy = mean(hwy),
  median_hwy = median(hwy)), keyby = .(manufacturer, cyl)]
```

When your done wrangling... Just collect and visualize

Use the `collect()` function or `as_tibble()` function to apply the data.table translation to your lazy data table and extract the results.

```

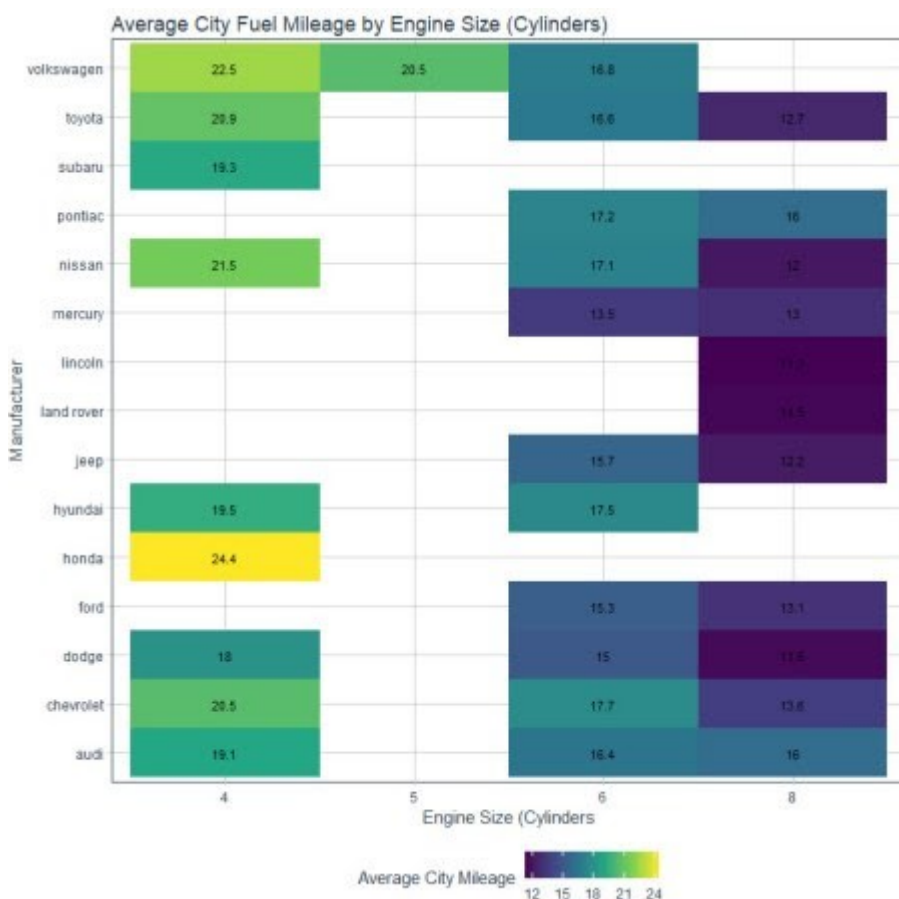
Collect and Visualize

32 # * Collect and Convert to Tibble
33 mpg_summary_tbl <- mpg_summary_dt %>% collect()
34
35 # 2.0 GGLOT ----
36
37 # * City Fuel Mileage Heat Map ----
38 mpg_summary_tbl %>%
39   ggplot(aes(manufacturer, factor(cyl), fill = mean_cty)) +
40   geom_tile() +
41   geom_text(aes(label = round(mean_cty, 1)), size = 3) +
42   scale_fill_viridis_c(direction = 1) +
43   labs(title = "Average City Fuel Mileage by Engine Size (Cylinders)",
44        x = "Manufacturer", y = "Engine Size (Cylinders)",
45        fill = "Average City Mileage") +
46   coord_flip() +
47   tidyquant::theme_tq()
48

```

The ggplot2 code produces this visualization. We can easily see:

- Honda has the highest City Mileage in small engine cars (24.4 MPG)
- Audi has the highest City Mileage in large engine cars (16 MPG)



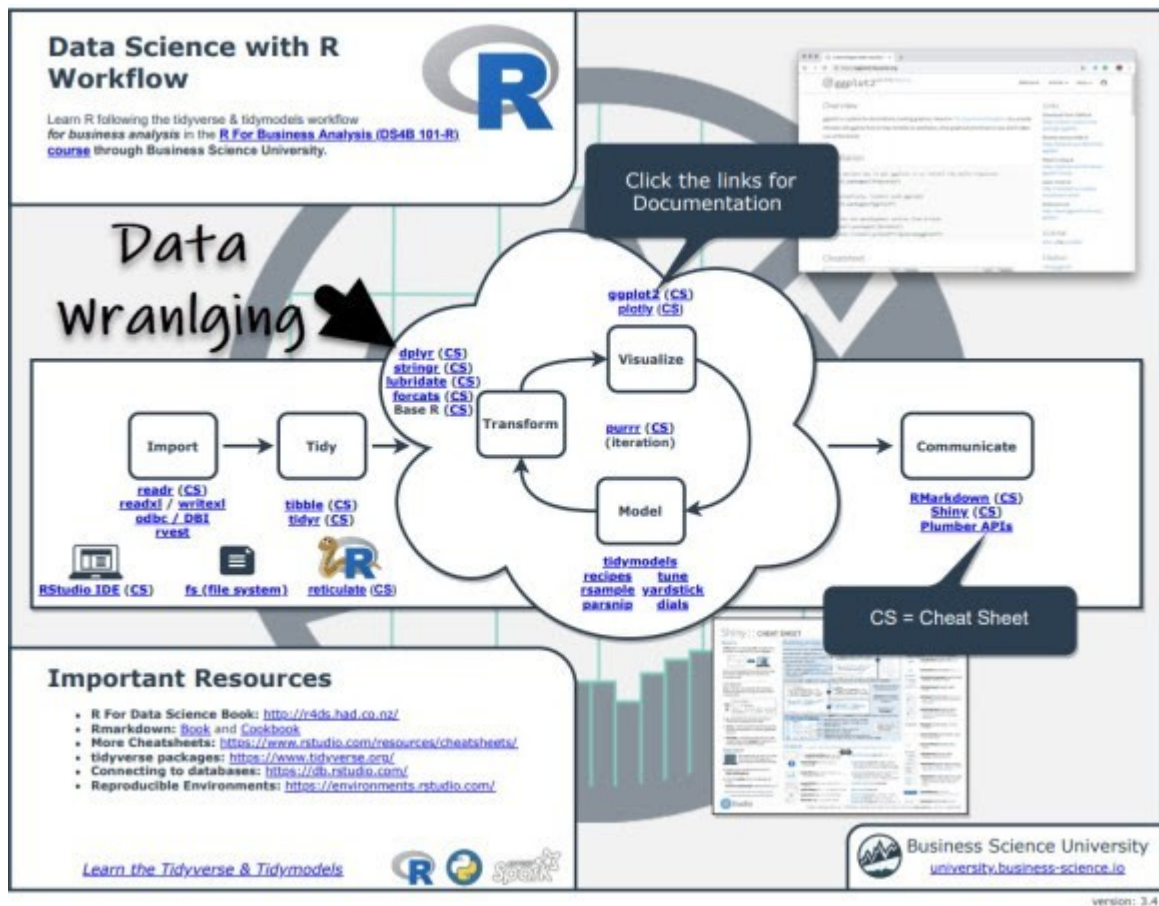
Learning Data Wrangling with Dplyr

It should be obvious now that **learning dplyr is insanely powerful**. Not only is it beginner-friendly, it unlocks `data.table`, the fastest in-memory data wrangling tool. Here are a few tips.

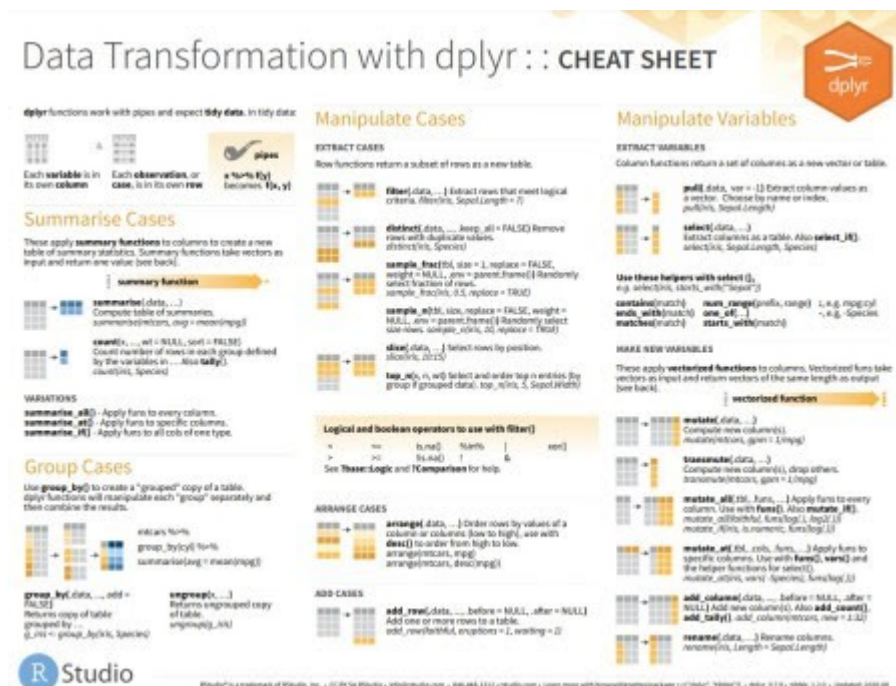
Pro Tip 1 – Use the Cheat Sheet

Dplyr is an 80/20 tool shown on the first page of my [Ultimate R Cheat Sheet](#).

Click the “CS” next to dplyr to get the Data Wrangling with Dplyr Cheat Sheet. Woohoo!



Clicking the “CS” opens the Data Transformation with Dplyr Cheat Sheet. Now you’re ready to begin learning Dplyr.



PRO TIP 2 – Learn Dplyr in my Business Analysis with R Course

It might be difficult to learn Dplyr on your own. I have a course that walks you through the entire process from analysis to reporting.

The **R for Business Analysis 101 Course** is the first course in my **R-Track program**. You'll do a

1. Customer Segmentation Report
2. Product Pricing Estimation and Gap Analysis

The image displays a Jupyter Notebook environment. The left pane shows Python code for a K-means clustering implementation. The code defines a `Customer` class with attributes `name`, `age`, and `income`, and methods for calculating distance and performing K-means clustering. The right pane shows a 'Customer Segmentation Report' dated '12/30/2018'. The report includes a 'Customer Segmentation Overview' section with a brief description of the report and a 'Results' section with a warning about the lack of guaranteed optimal clustering. A scatter plot titled 'Visualizing Clusters' shows data points colored by cluster, with labels for 'Cluster 0 (Low)', 'Cluster 1 (Mid)', and 'Cluster 2 (High)'.

You just sliced and diced data with `dtplyr` – the `data.table` backend to `dplyr`.