R bootcamp - part 3: compound figures with cowplot

Hannah Meyer

September 2025

Contents

1 Background

There are many add on packages for `ggplot2`. Here, I want to highlight and show examples for one of those packages: `cowplot` (short for **C**laus **O**. **W**ilkes plot package, initially developed for Claus's lab members). It is extensively documented on this webpage: cowplot.

`cowplot` is described as

```
provid[ing] various features that help with creating publication-
quality figures, such as a set of themes, functions to align plots and
arrange them into complex compound figures, and functions that make it easy
to annotate plots and or mix plots with images.
```

2 Setting up

In the following two chunks, we will again set up our analys document by specifying the options for `knitr` and

```
knitr::opts_chunk$set(echo = TRUE,
                      comment = "#>",
                      collapse = TRUE,
                      fig.width = 6,
                      fig.align = "center",
                      out.width = "70%")
```

load all libraries required for our analysis:

```
library("readr")
library("ggplot2")
library("cowplot")
library("RColorBrewer")
```

3 Individual visualisations

In the following section, we will load the dataset we worked with in the previous session and recreate the visualisations we used to show the antigenic maps, histogram and boxplots of the clusters by time.

Unlike the previous session, where we immediately displayed the output of each `ggplot2` call, we will now save each plot into a different object. To do so, we will first create the common `ggplot` object `p`, that relies on our dataset `coord`:

```
coord <- read_csv("data/2004_Science_Smith_data.csv")
#> Rows: 322 Columns: 9
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> chr (4): name, cluster, type, location
#> dbl (5): year, x.coordinate, y.coordinate, lat, lng
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
p <- ggplot(data=coord)
```

To recreate the previous plots, we save them into a new object by adding the appropriate layers to `p`:

```
antigenic_map <- p +
  geom_point(aes(x=x.coordinate, y=y.coordinate, color=cluster)) +
  scale_color_brewer(type="qual", palette = "Set3") +
  labs(x="Dimension 1 [AU]",
       y="Dimension 2 [AU]",
       color="Cluster") +
  theme_bw()

time_barplot <- p + geom_bar(aes(x=year, fill=cluster),
                             position=position_dodge(preserve="single")) +
  scale_fill_brewer(type="qual", palette = "Set3") +
  labs(x="Count",
       y="Year",
       color="Cluster") +
```

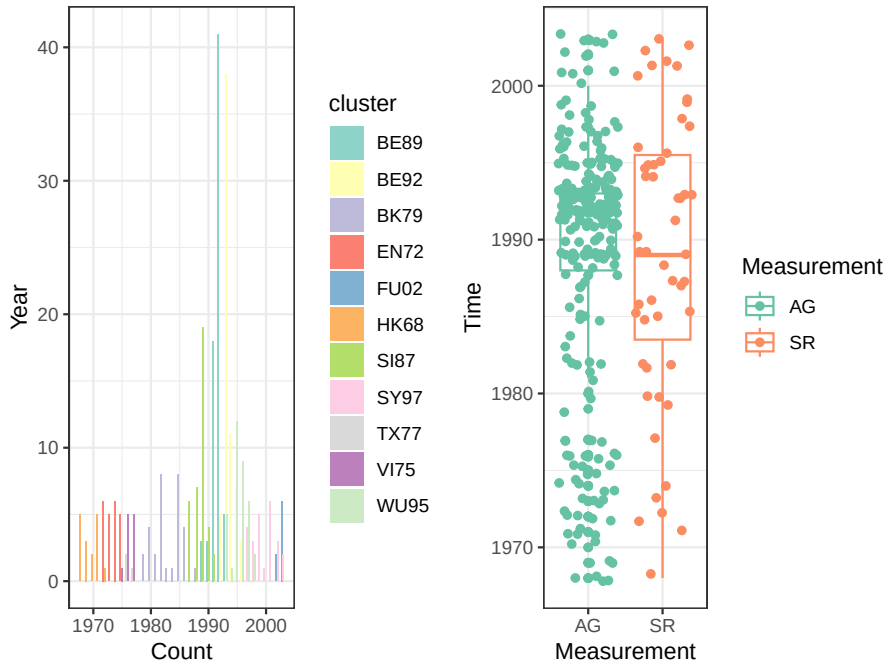
```
  theme_bw()

time_boxplot <- p +
  geom_boxplot(aes(x=type, y=year, color=type)) +
  geom_jitter(aes(x=type, y=year, color=type)) +
  scale_color_manual(values=c("#66c2a5", "#fc8d62")) +
  labs(x="Measurement",
       y="Time",
       color="Measurement") +
  theme_bw()
```

`cowplot`'s `plot_grid` function takes these plotting objects as input and arranges them into a grid.

4 Simple compound plots

To generate our first compound figure, we provide the bar and boxplot objects to `plot_grid`, which arranges them in a single row, next to one another.

```
plot_grid(time_barplot, time_boxplot)
```



4.1 Customising `plot_grid`

`plot_grid` is highly customisable, including layout, labels, alignments and scaling.

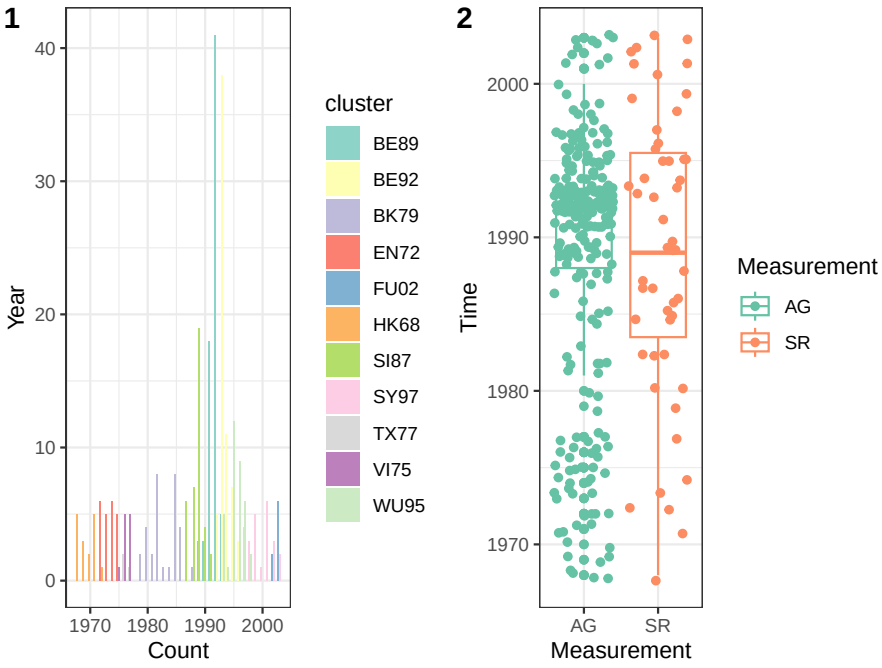
4.1.1 Labels

To create publication-ready figures, we can set a label for each plot, by either specifying the desired labels in a vector of the same lengths as the number of plots (as we do here for 2 plots) or by setting `labels="AUTO"` to auto-generate upper-case labels or `labels="auto"` to auto-generate lower-case labels.

Reminder: vectors are created with `c()`, by providing its elements separated by commons, e.g. `c(1.6, 2.5, 3.2)` is a vector with three elements of the type double.

- labels have to be provided for all or none of the panels i.e. the length of the vector has to be the same as number of panels
- to label only certain panels make use of the empty string: `""` e.g. `c("a", "")` will label first panel with a, second panel with nothing

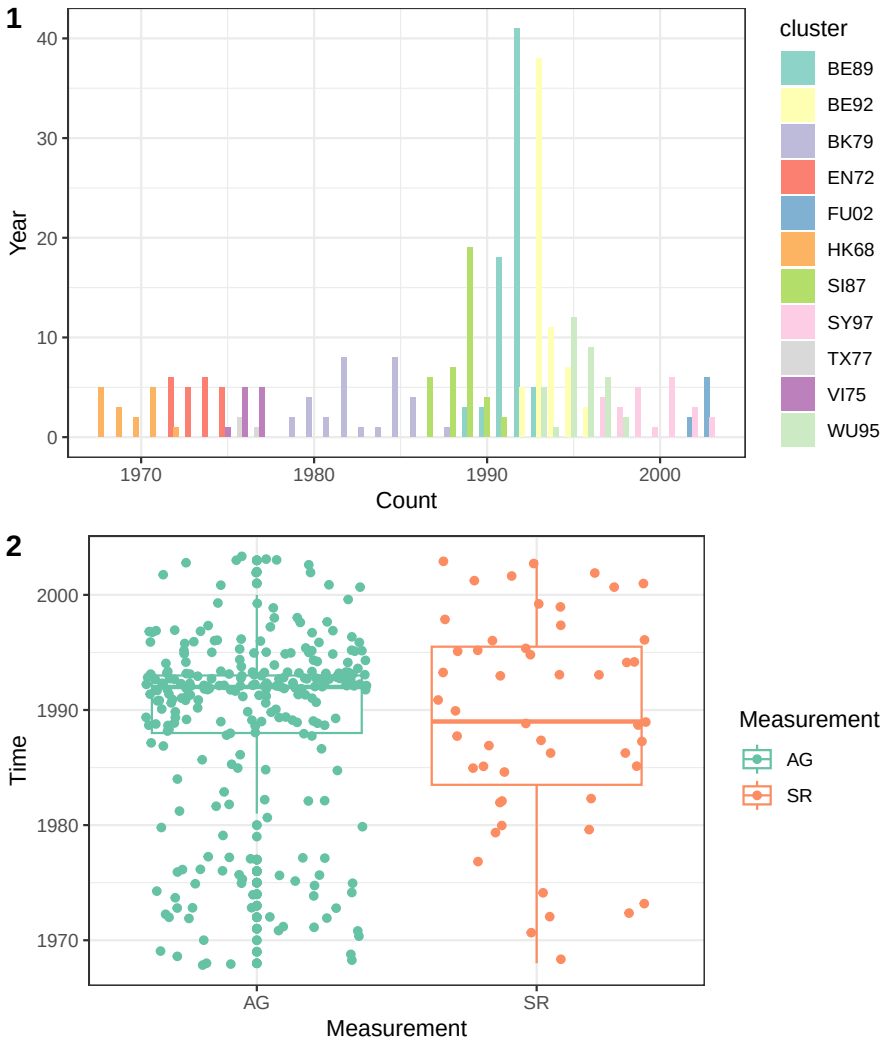
```
plot_grid(time_barplot, time_boxplot,
          labels=c(1,2))
```



4.1.2 Layout

Per default, `plot_grid` chooses a 'one row, two column' layout for the plots we specified. To specify a 'two row, one column' layout we can either specify the number of colums `ncol` or number or rows `nrow` argument:

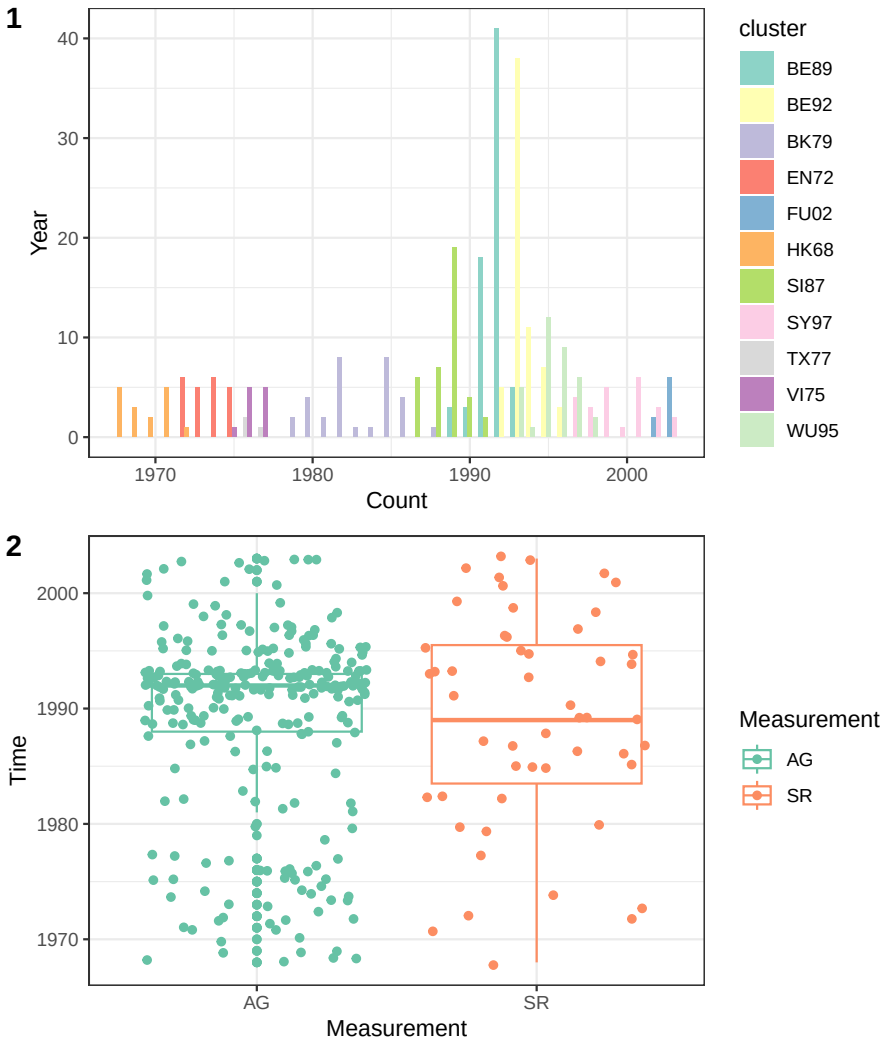
```
plot_grid(time_barplot, time_boxplot,
          labels=c(1,2),
          nrow=2)
```



4.1.3 Alignment

You might notice that in the one column layout, the plots are slightly shifted with respect to their y-axis and legends alignment. Per default, `plot_grid` keeps the *axis titles* aligned. To align by *actual axis*, we can use a combination of the `align` and `axis` arguments. Here we want the plots to be vertically aligned along the left and right margins of the plot panel:

```
plot_grid(time_barplot, time_boxplot,
          labels=c(1,2),
          nrow=2,
          align="v",
          axis="lr")
```



4.2 Exercises

1. Autogenerate lower-case labels for the compound plot of `time_boxplot` and `time_barplot`.
2. Change the default layout by specifying the number of columns.
3. Read the help function for `plot_grid` and experiment with other `axis` options.

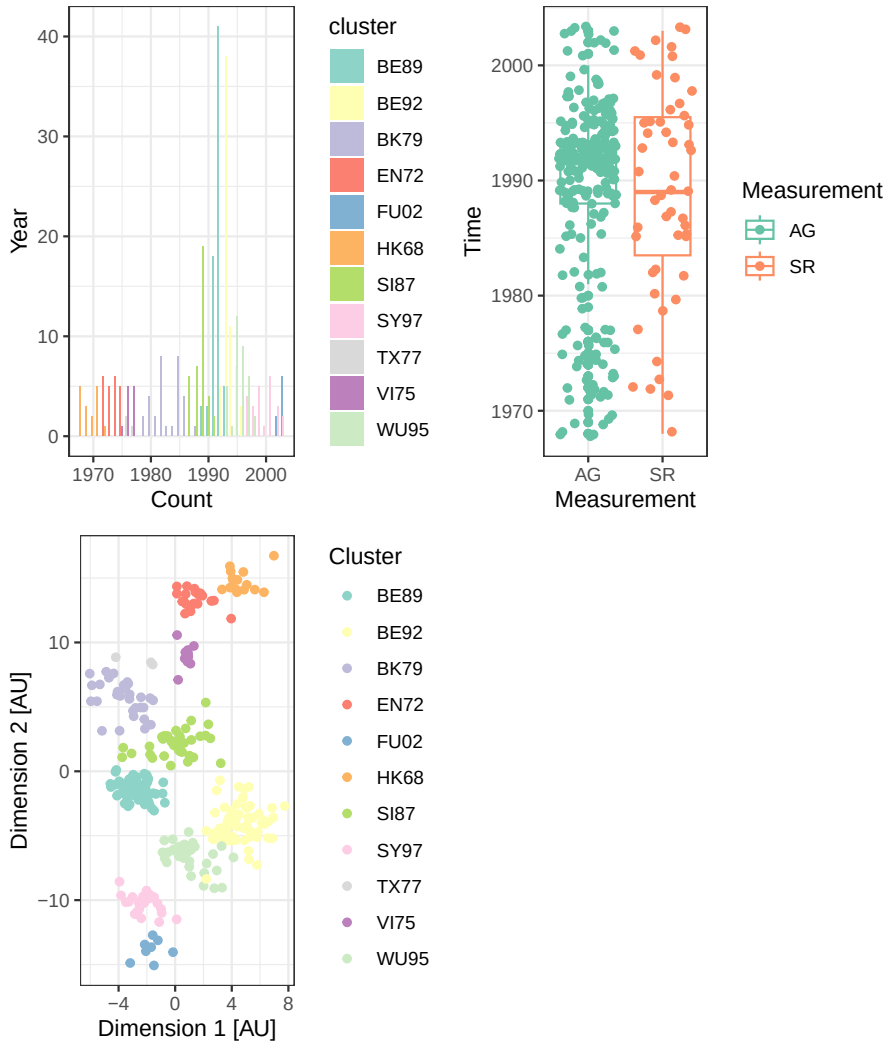
5 Nested compound plots

So far, we have only considered the two time-related plots we generated to visualise the antigenetic cluster data. In the following few chunks, we will see how we can add a third plot to the grid that has different dimensions than then previous two.

5.1 Arranging plots

Let's start with the default behaviour:

```
plot_grid(time_barplot, time_boxplot, antigenic_map)
```



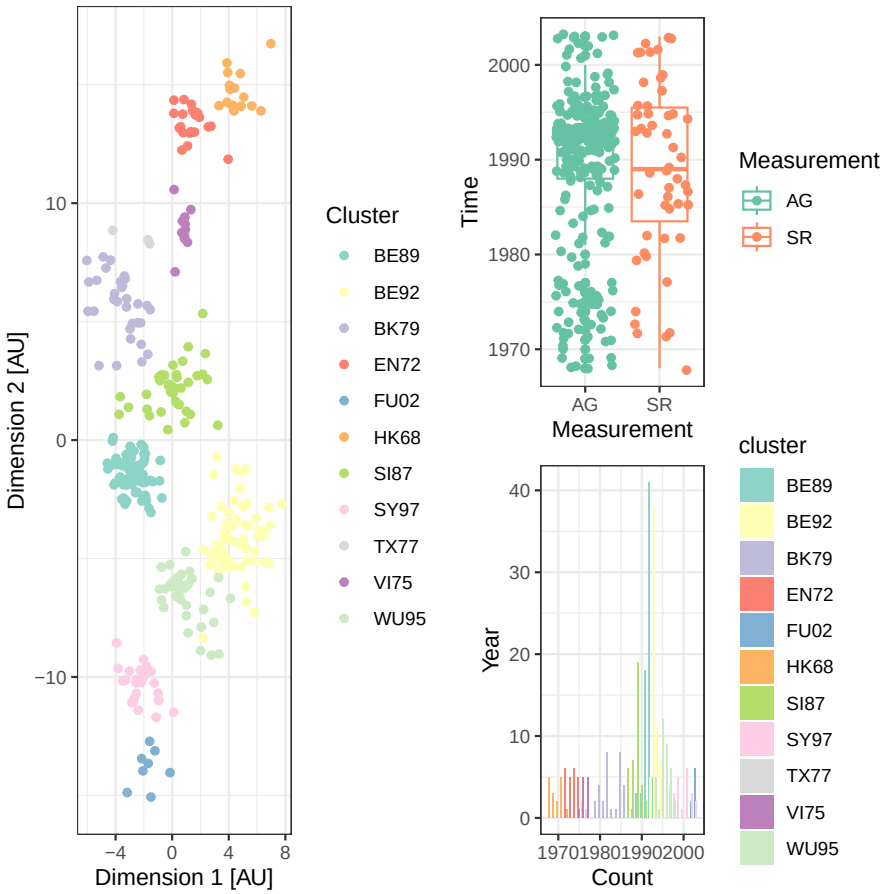
This layout does not look great, for several reason:

1. We have a void in the lower right corner;
2. The heights of both rows are the same, which unnessecarily stretches the bar and boxplot.

To address both issues, we can arrange a two-column layout, with the first column containing the antigenic map only, spanning over two rows, and the bar and boxplot depicted in the first and second row of the second column, respectively.

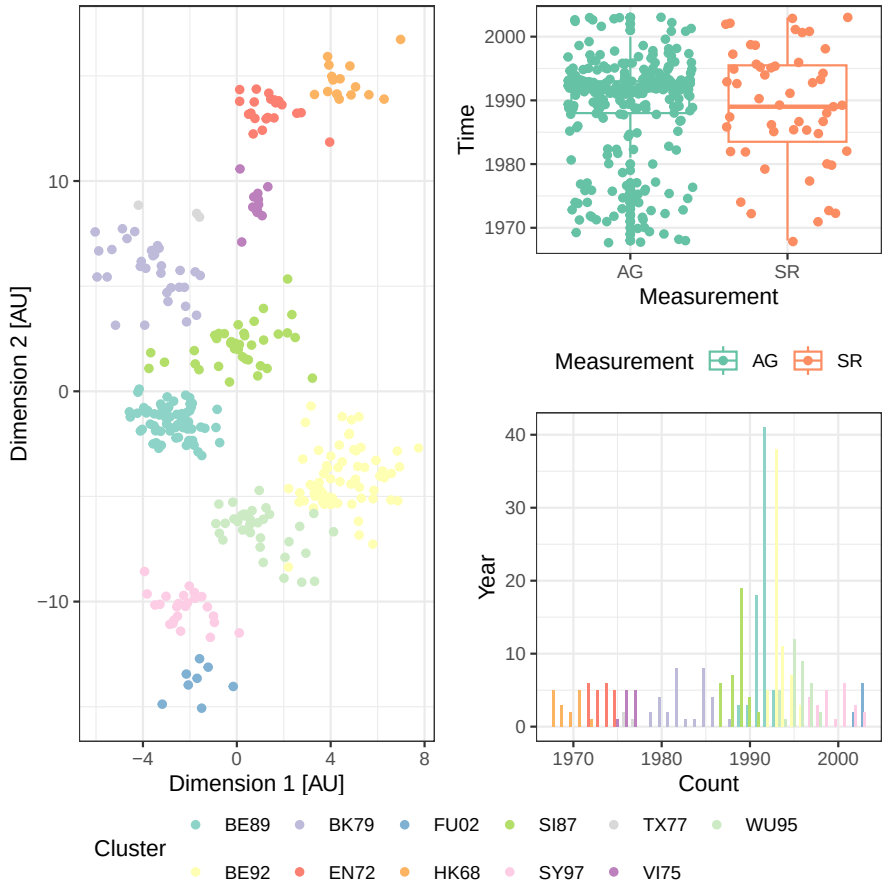
To achieve this, we will use two calls to `plot_grid`, the first to generate our composite for the right-hand column, the second to put this composite together with the antigenic map.

```
right_column <- plot_grid(time_boxplot, time_barplot,
                  nrow=2,
                  align="v",
                  axis="lr")

plot_grid(antigenic_map, right_column,
          nrow=1,
          align="h",
          axis="t")
```



5.2 Shared legends

In the visualisation above, we have color-coded both the antigenic map and the bar plot by cluster. Below, we will have a look how we can add one shared legend as an additional row to `plot_grid` and remove the individual legends. This is what our final figure will look like:



Let's have a look at the steps required to get there:

First, we will create the individual components of our panel, this time adding `legend.position` via `theme`. We create the `right_column`, similar as above, but, in addition, specify that in this `plot_grid`, the boxplot legend should move below the plot (`theme(legend.position = "bottom")`) and the legend from the barplot should be removed with `theme(legend.position = "none")`. Similarily, we remove the the legend from the `antigenic_map` that we pass to `plot_grid`.

Note: We only move and remove the legends from the objects that we pass to `plot_grid`, the original objects remain unchanged.

```
right_column <- plot_grid( time_boxplot + theme(legend.position = "bottom"),
                           time_barplot + theme(legend.position = "none"),
                           nrow=2,
                           align="v",
                           axis="lr")

top_row <- plot_grid(antigenic_map + theme(legend.position = "none"),
                     right_column, nrow=1)
```

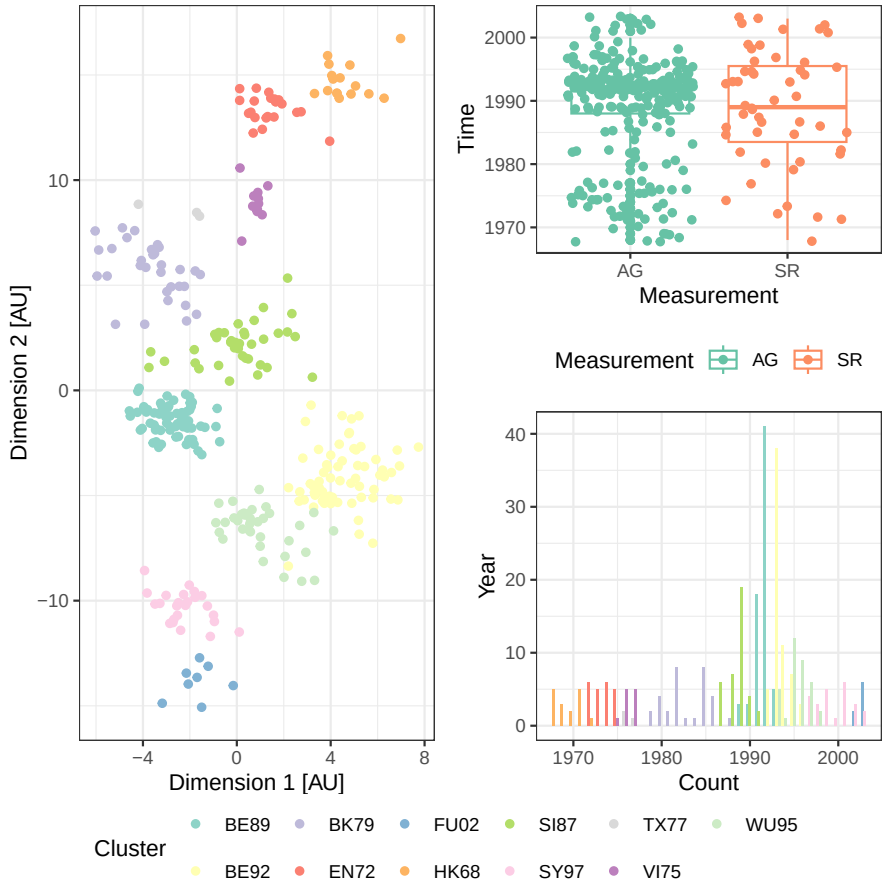
We then create a new map object from which we extract the legend. We will place the shared legend as an additional row at the bottom of the our panel plot, so we want a legend with horizontal layout, as if it was at the bottom of a plot. Thus, we specify in this new object, that the legend should be at the bottom of the plot (as we saw above) and, in addition, specify that the color legend should have two rows for its elements with `guides(color=guide_legend(nrow=2))`.

We then extract the legend from the `new_map` object with `get_legend`:

```
new_map <- antigenic_map +
  guides(color=guide_legend(nrow=2)) +
  theme(legend.position = "bottom")

cluster_legend <- get_legend(new_map)
```

Finally, we will put it all together in a new call to `plot_grid`. In addition to the arguments that we have already seen for `plot_grid`, we also specify `rel_heights` here, which says that the ratio of the first row (i.e. our compound panel) to the second row (the shared legend) should be 10:1.

```
plot_grid(top_row, cluster_legend,
          nrow=2,
          rel_heights = c(10,1))
```



5.2.1 Exercises

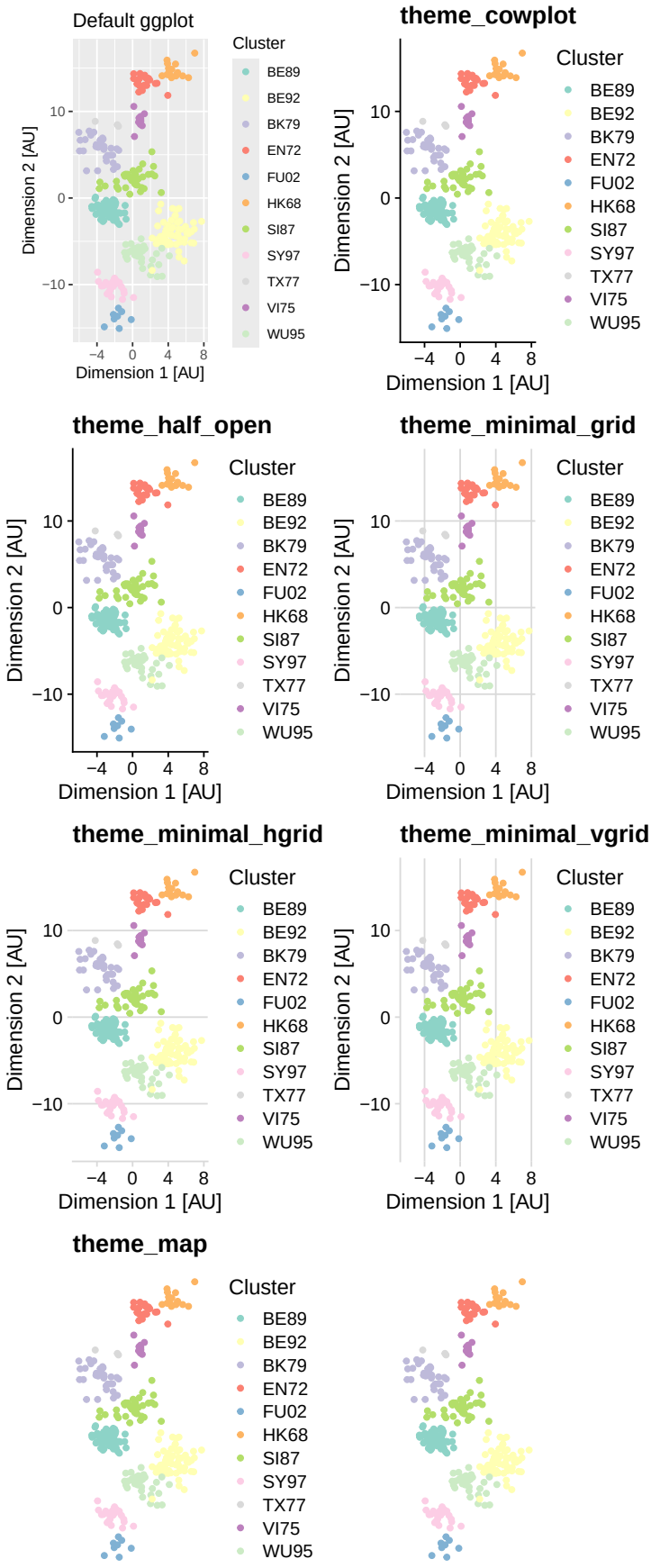
1. What would be a good labeling strategy for this nested plot? Label accordingly.
2. Play with the ratios provided to `rel_heights`. Why does it not make sense to provide `rel_widths` in this example?

6 `cowplot` beyond `plot_grid`

Many people come to `cowplot` in the search of a function for composite figures (`plot_grid`). `cowplot` has other things to offer, most notably for me, it extends the range of different `themes` provided by `ggplot`.

Below an overview of available cowplot themes, all applied to the `antigenic_map` scatter plot:

```
plot_grid(antigenic_map + theme_gray() + ggtitle("Default ggplot"),
          antigenic_map + theme_cowplot()  + ggtitle("theme_cowplot"),
          antigenic_map + theme_half_open()  + ggtitle("theme_half_open"),
          antigenic_map + theme_minimal_grid() + ggtitle("theme_minimal_grid"),
          antigenic_map + theme_minimal_hgrid() + ggtitle("theme_minimal_hgrid"),
          antigenic_map + theme_minimal_vgrid() + ggtitle("theme_minimal_vgrid"),
          antigenic_map + theme_map() + ggtitle("theme_map"),
          antigenic_map + theme_nothing() + ggtitle("theme_nothing"),
          nrow=4,
          align="vh",
          axis="tblr")
```


Default ggplot
theme_cowplot
theme_half_open
theme_minimal_grid
theme_minimal_hgrid
theme_minimal_vgrid
theme_map
Cluster
BE89
BE92
BK79
EN72
FU02
HK68
SI87
SY97
TX77
VI75
WU95
Dimension 1 [AU]
Dimension 2 [AU]
−4
0
4
8
10
0
−10

All themes are `cowplot` themes apart from the default `ggplot2` theme shown for reference. The lower right plot shows `theme_nothing` by `cowplot`.

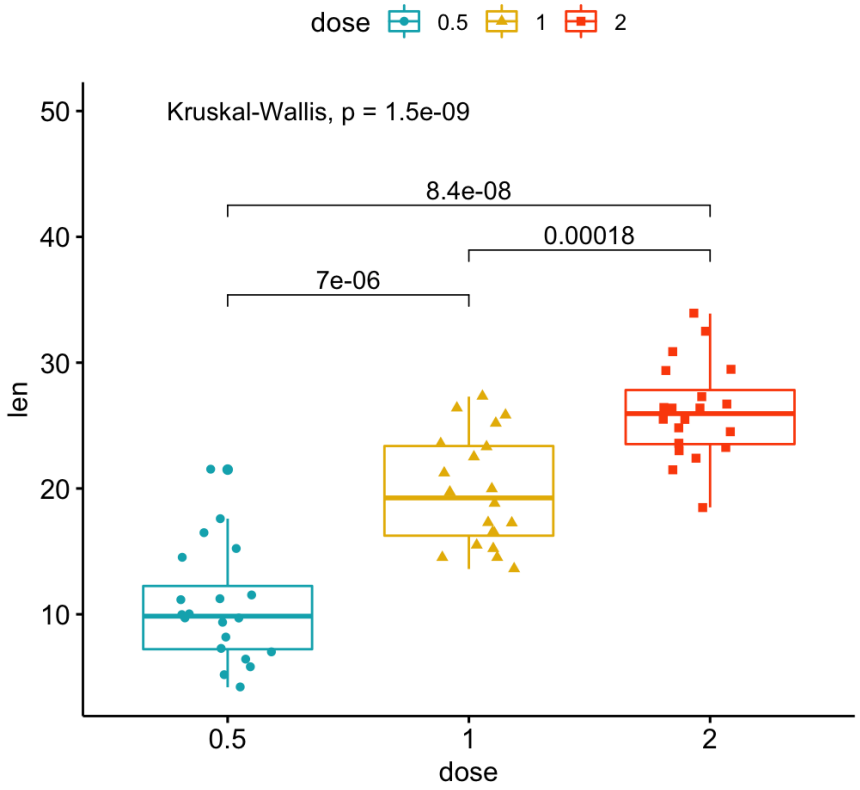
Note: Not all themes are equally well suited for each plot type. The `ggplot2` default for boxplots chooses a simply horizontal grid for instance, where as the same theme applied to the scatter plot display horizontal and vertical grids.

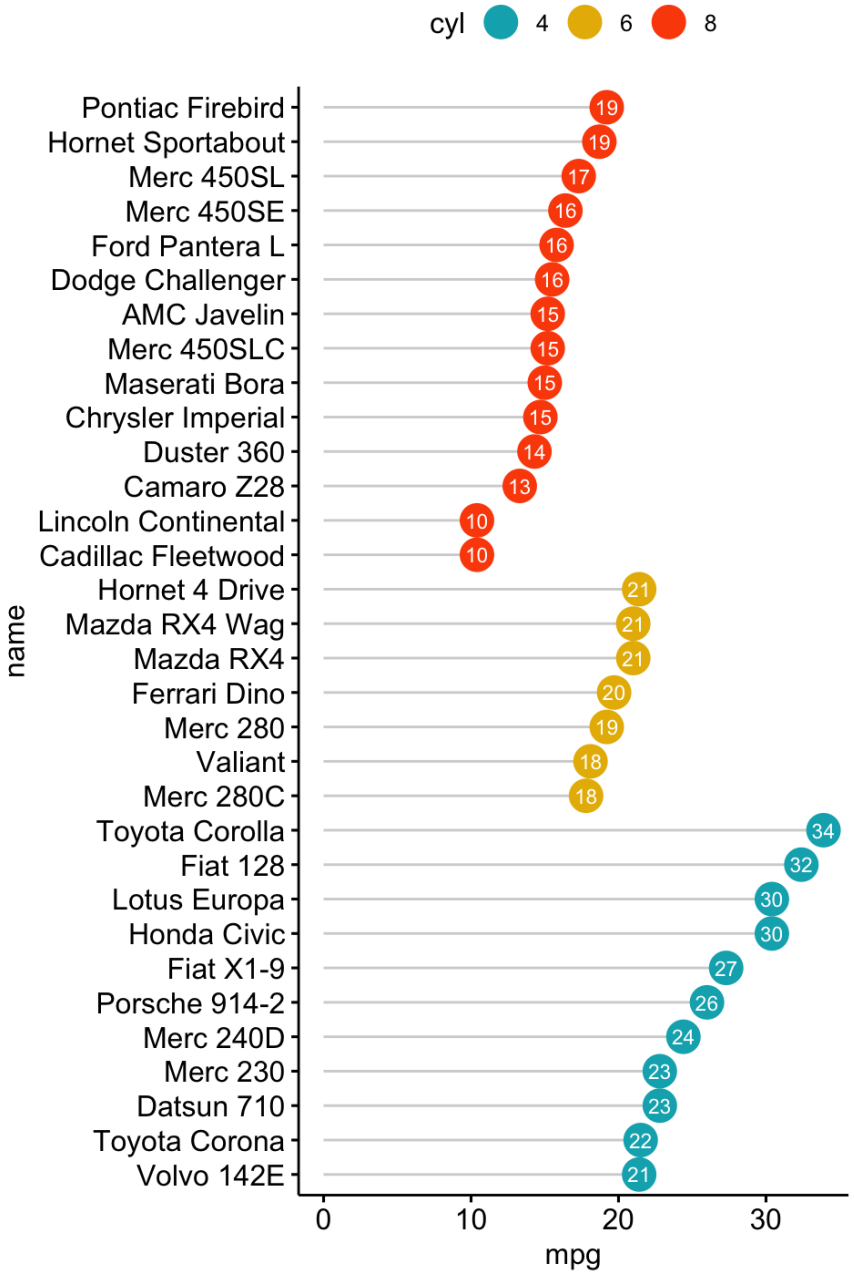
6.1 Exercises

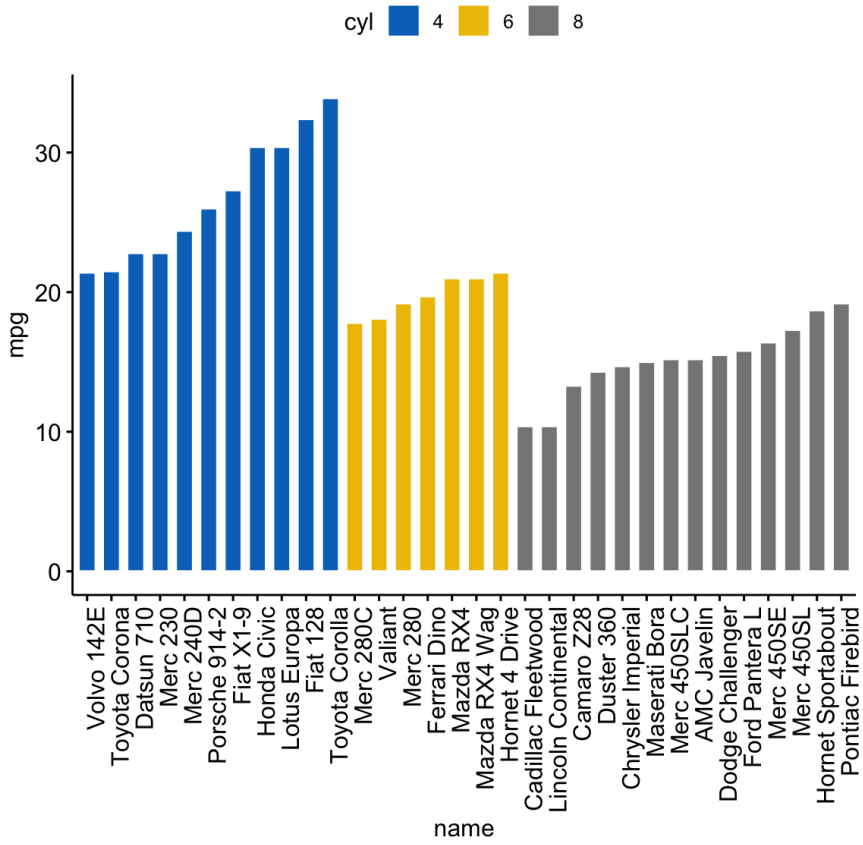
1. Discuss which other plot type might benefit from a simpler grid than the default one chosen?
2. What happens if you add a theme to `plot_grid`?
3. Apply different themes to the three individual plots and visualise in `plot_grid`.

7 Additional material

The `ggpubr` package provides some easy-to-use functions for creating and customizing `ggplot2`- based publication ready plots: ggplot2 Based Publication Ready Plots




cyl
4
6
8
Pontiac Firebird
Hornet Sportabout
Merc 450SL
Merc 450SE
Ford Pantera L
Dodge Challenger
AMC Javelin
Merc 450SLC
Maserati Bora
Chrysler Imperial
Duster 360
Camaro Z28
Lincoln Continental
Cadillac Fleetwood
Hornet 4 Drive
Mazda RX4 Wag
Mazda RX4
Ferrari Dino
Merc 280
Valiant
Merc 280C
Toyota Corolla
Fiat 128
Lotus Europa
Honda Civic
Fiat X1-9
Porsche 914-2
Merc 240D
Merc 230
Datsun 710
Toyota Corona
Volvo 142E
19
19
17
16
16
16
15
15
15
15
14
13
10
10
21
21
21
20
19
18
18
34
32
30
30
27
26
24
23
23
22
21
name
0
10
20
30
mpg



8 References

Cowplot documentation on this webpage: cowplot