Preparing homeworks and exams in STAT 961

Eugene Katsevich

August 24, 2021

1 Compilation

Compile your `Rnw` file to PDF by pressing the `Compile PDF` button or using a keyboard shortcut (e.g. `Command-Shift-K` on Mac). It is convenient to place RStudio and the compiled PDF in side-by-side windows on your computer as you work.

You may run into compilation issues for a variety of reasons. Here are a few trouble-shooting tips:

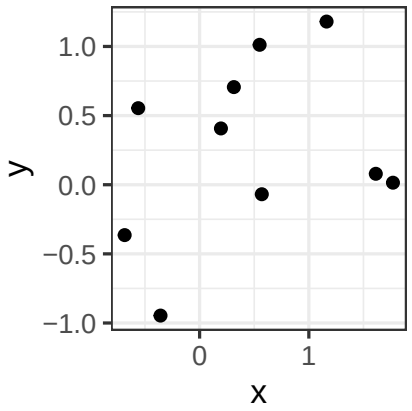
- Make sure you have followed all the steps in getting-started.pdf. This document may have been updated since you last saw it.
- You might be missing necessary `R` packages. Install these using `install.packages`.
- You may find that your PDF is not updating even though you have updated your `Rnw` file. The reason for this might have to do with caching, `knitr`'s behind-the-scenes efforts to save the outputs of code chunks to avoid re-running them. In certain cases, chunks might not be re-run even though they should be (you can read more about this here.) To avoid this issue, make sure you save any plots or tables in the same code chunk you generate them. If you've done this and the problem persists, simply delete the `cache` folder and re-compile.
- Your `R` code may have bugs. Usually the error message will point you to a line number where the code broke. Debug your code by stepping through it line-by-line interactively before compiling your report.
- Your LaTeX code may have bugs. Usually the error message will point you to a line number where the code broke.
- If you are stuck, post on Piazza or come to office hours and the teaching staff will assist you.

2 Adding figures and tables to your report

2.1 Figures

Suppose you write a chunk of `R` code that makes a figure:

```
library(tidyverse)
test_data = tibble(x = rnorm(10), y = rnorm(10))
p =  test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
plot(p)
```



While the figure is displayed in your report, it comes without a caption or a label you can use to reference the figure in your text. Instead of simply plotting the figure in your code, save the figure using `ggsave` and then include it in your LaTeX code using `\includegraphics`:

```
# create figure but don't plot it
p = test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
# save the figure
ggsave(plot = p, filename = "test_plot.png",
       device = "png", width = 2, height = 2)
```

```
% plot the figure using LaTeX
\begin{figure}[h!]
\centering
\includegraphics[width = 0.4\textwidth]{test_plot.png}
\caption{This is a test plot.}
\label{fig:test-plot}
\end{figure}
```

This will give you Figure 1, which you can then reference in your LaTeX code using `\ref{fig:test-plot}`.

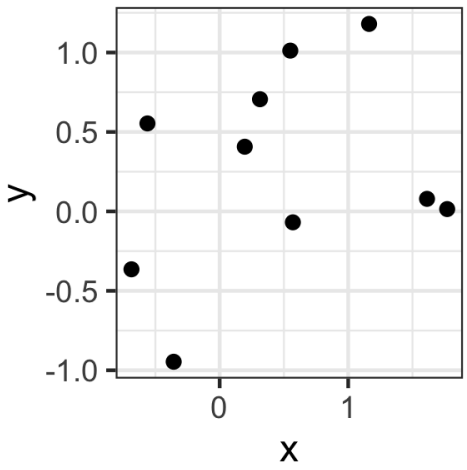


Figure 1: This is a test plot.

2.2 Tables

Creating tables follows the same paradigm, except with `kableExtra::kable` and `kableExtra::save_kable` replacing `ggplot` and `ggsave`, respectively. For example:

```
# run a regression (or do something else to produce a table)
lm_fit = lm(y ~ x, data = test_data)

# create and save table
coef(summary(lm_fit)) %>%
  kableExtra::kable(format = "latex", row.names = NA,
                    booktabs = TRUE, digits = 2) %>%
  kableExtra::save_kable("test_table.png")
```

```
% add table to report via LaTeX
\begin{table}[h!]
\centering
\includegraphics[width = 0.6\textwidth]{test_table.png}
\caption{Test table}
\label{tab:test-table}
\end{table}
```

This will give you Table 1, which you can then reference in your LaTeX code using `\ref{tab:test-table}`.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.16	0.24	0.67	0.52
x	0.22	0.25	0.85	0.42

Table 1: Test table

3 High-quality reports

Aside from statistical methodology and computing, another goal of STAT 961 is to teach you how to produce high-quality reports. This skill is essential to successfully communicating the results of your research, e.g. in the form of a manuscript submitted for publication. Therefore, each submitted homework and exam will be held to a high standard of presentation, which will be evaluated and will comprise a small part of your grade. Below are guidelines on producing high-quality reports, broken down by their components: text, code, figures, and tables.

3.1 Text

Your prose should be clear and concise. Use references to refer to equations, figures, and tables.

3.2 Code

Your code should be commented and easy to read. Make sure that your code does not exceed the width of the page, like this:

```
# a line that exceeds the width of the page
tibble(x = 1:100, y = 5*x + rnorm(100, sd = 100)) %>% filter(x < 80) %>% summarise(sample_corre

## # A tibble: 1 x 1
##   sample_correlation
##                <dbl>
## 1              0.801
```

To avoid such long lines of code, make sure your code does not reach the vertical line in the right-hand side of your RStudio editor. Insert line breaks appropriately to make your code more readable:

```
# appropriate line breaks added
tibble(x = 1:100, y = 5*x + rnorm(100, sd = 100)) %>%  # generate data
  filter(x < 80) %>%                                   # subset data
  summarise(sample_correlation = cor(x, y))            # evaluate sample corr.

## # A tibble: 1 x 1
##   sample_correlation
##                <dbl>
## 1              0.794
```

3.3 Figures

Figures are very important tools to convey information to readers, and they should be constructed thoughtfully. Please read Chapter 28 of R for Data Science, which is a good reference for producing high-quality figures. Here we discuss some of the most important elements.

Sizing. The **aspect ratio** (i.e. ratio of width to height) of your plots is consistent with their content; e.g. box plots are usually relatively narrow, and scatter plots often make sense with equal aspect ratios.

The **absolute size** of your figures (passed to `ggsave` via the `width` and `height` arguments) should be such that the the text on the plot is easy to read. Consider the following three choices for the absolute sizes of the test plot from Figure 1:

```
# small
ggsave(plot = p, filename = "test_plot_small.png",
       device = "png", width = 1, height = 1)
# medium
ggsave(plot = p, filename = "test_plot_medium.png",
       device = "png", width = 2, height = 2)
# large
ggsave(plot = p, filename = "test_plot_large.png",
       device = "png", width = 5, height = 5)
```

These three choices are compared in Figure 2; the small-sized plot is too cramped, the large-sized plot has axis titles and labels that are too small to read, and the medium-sized plot is about right. A good rule of thumb is that the smallest text in your plots should be roughly the same size as the text in your report.

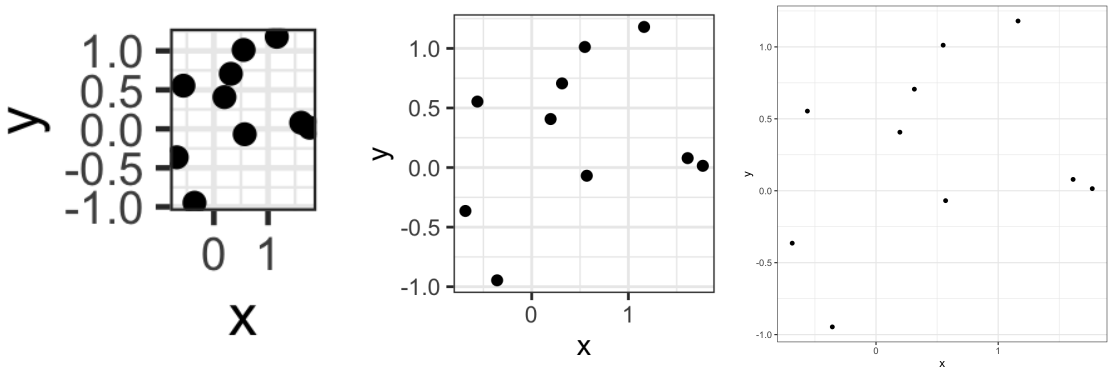


Figure 2: The plot from Figure 1 saved as three different absolute sizes (small, medium, and large).

The **relative size** of your figures (relative to the dimensions of your report, as specified in the `\includegraphics` command) should also be chosen appropriately. Compare Figures 3, 4, and 5, which correspond to the following three relative sizes:

```
% small relative size
\includegraphics[width = 0.1\textwidth]{test_plot.png}

% medium relative size
\includegraphics[width = 0.4\textwidth]{test_plot.png}

% large relative size
\includegraphics[width = 0.8\textwidth]{test_plot.png}
```

The small plot is too small to see, the large plot takes up too much space, and the medium one is about right.

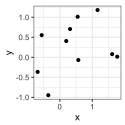


Figure 3: The plot from Figure 1 included with `width = 0.1\textwidth`. Its relative size is too small.

Titles. Each plot should include informative axis and legend titles. For example, consider the code below (drawn from R4DS Chapter 28), which produces the plot in Figure 6.

```r
# a plot without clear axis and legend titles
p = mpg %>%
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme_bw()
```

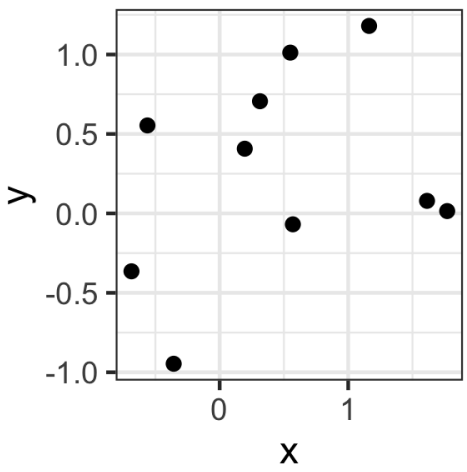


Figure 4: The plot from Figure 1 included with `width = 0.4\textwidth`. Its relative size is about right.

```
# save plot
ggsave(plot = p, filename = "cars-unlabeled.png",
       device = "png", width = 5, height = 3.75)
```

This is a plot of fuel efficiency versus engine displacement for various types of cars, but the axis and legend labels on the plot do not make this very clear. We can easily add informative titles to this plot using `labs`, resulting in Figure 7, which is much easier to understand.

```
# a plot with clear axis and legend titles
p = mpg %>%
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (liters)",
    y = "Highway fuel economy (miles per gallon)",
    colour = "Car type"
  ) +
  theme_bw()

# save the plot
ggsave(plot = p, filename = "cars-labeled.png",
       device = "png", width = 5, height = 3.75)
```

Plots might or might not need overall titles; often the axis titles speak for themselves and the message of the plot can be conveyed in the caption (as in Figure 7.) To add plot titles if necessary, use `ggtitle`.

If applicable, axis titles should also include the units of measurement, e.g. liters or miles per gallon as in Figure 7. If axis titles involve mathematical formulas, these should be typeset appropriately. The code below (drawn from R4DS Chapter 28) and Figure 8, which it produces,

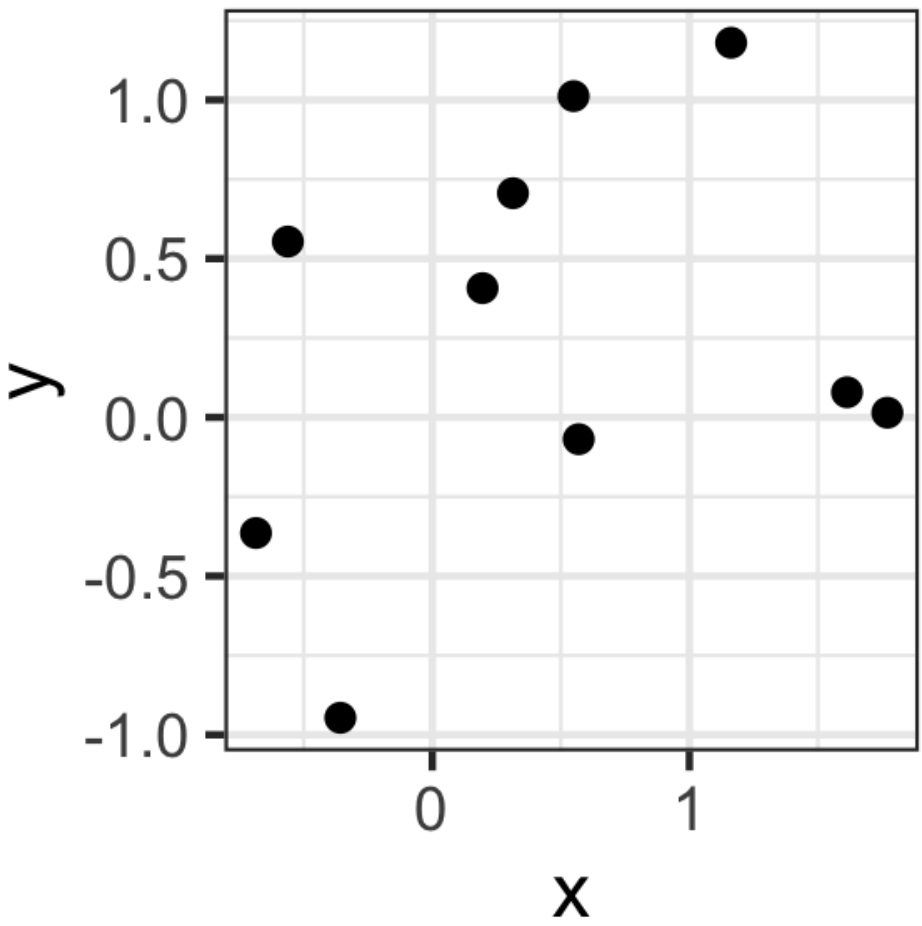


Figure 5: The plot from Figure 1 included with `width = 0.8\textwidth`. Its relative size is too large.

illustrate how to do this. More examples can be found at `?plotmath`.

```
# a plot illustrating how to include formulas in axis titles
p = tibble(x = runif(10),
       y = runif(10)) %>%
  ggplot(aes(x, y)) +
  geom_point() +
  labs(x = quote(sum(x[i] ^ 2, i == 1, n)),
       y = quote(alpha + beta + frac(delta, theta))) +
  theme_bw()

# save the plot
ggsave(plot = p, filename = "fig-formulas.png",
       device = "png", width = 2.5, height = 2.5)
```

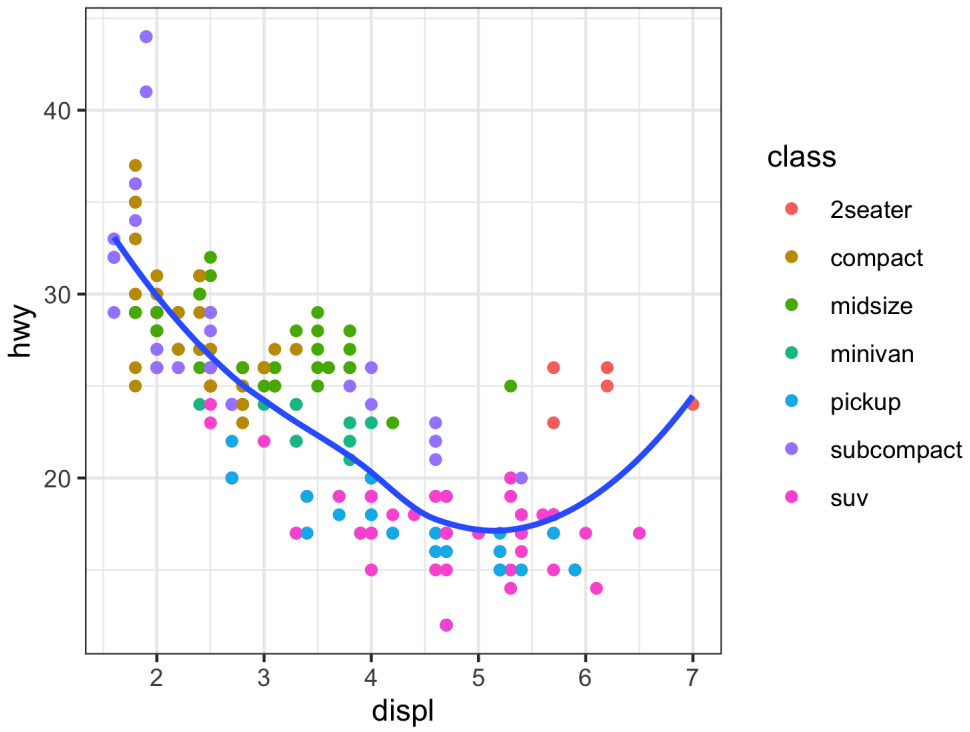


Figure 6: A plot without clear titles.

Captions. Figures should have informative captions to help readers understand what information is displayed and how to interpret it.

Layout. Sometimes, two or more plots make sense to present together in a single figure. This can be accomplished in two ways. If the different plots convey the same type of information but for different slices of the data, then `facet_grid` and `facet_wrap` are the best way of laying out these plots. For example, the code below and Figure 9 illustrates `facet_wrap` for the `mpg` data used in Figures 6 and 7.

```
# illustrate how to use facet_wrap to create a multi-panel plot
p = mpg %>%
  filter(class %in%
           c("2seater", "compact", "midsize")) %>%  # select 3 classes of cars
  ggplot(aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(class ~ .) +                           # separate panels per class
  labs(
    x = "Engine displacement (liters)",
    y = "Highway fuel economy\n(miles per gallon)", # line break in axis title
  ) +
  theme_bw()

# save the plot
ggsave(plot = p, filename = "facet-wrap.png",
```

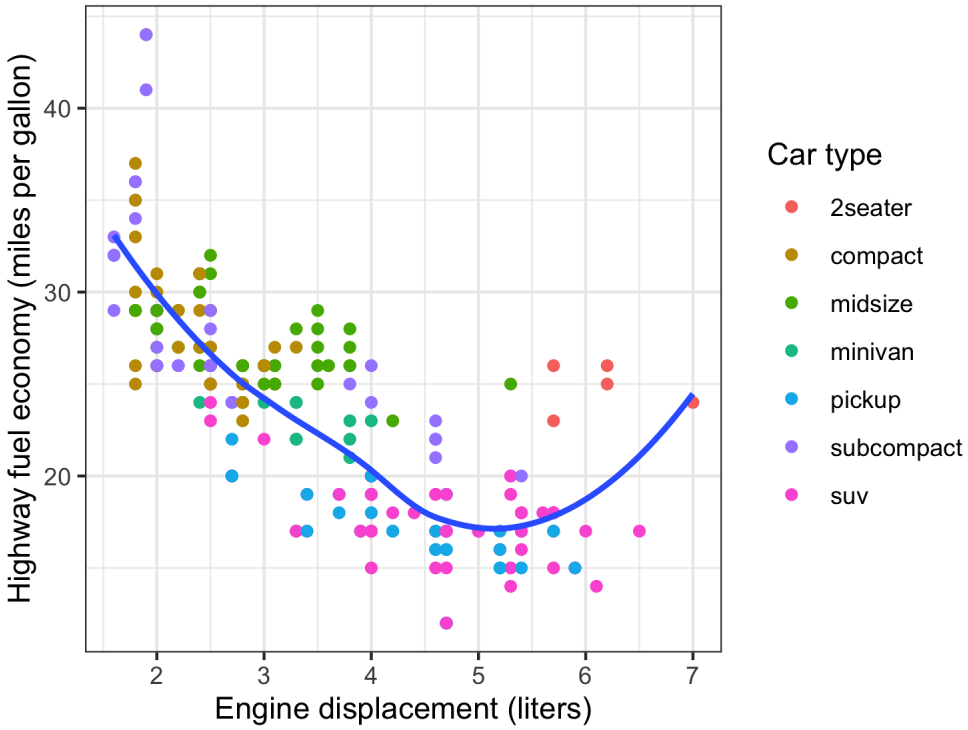


Figure 7: (A plot with clear axis and legend titles). Fuel efficiency generally decreases with engine size; two-seaters (sports cars) are an exception because of their light weight.

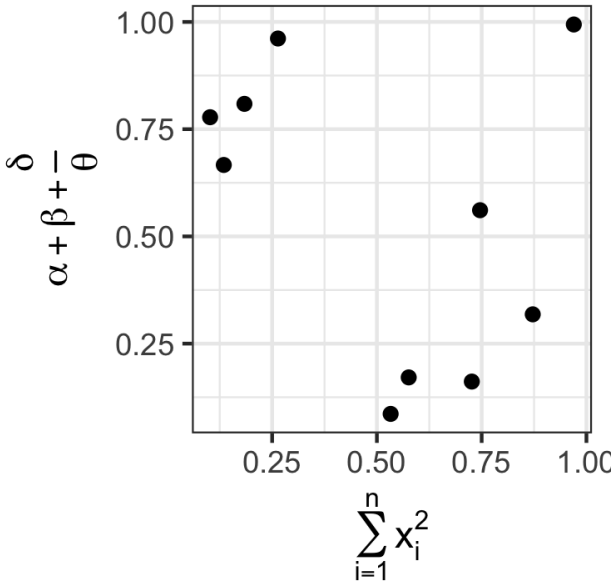


Figure 8: An illustration of using formulas in axis titles.

```
      device = "png", width = 5.5, height = 2.25)
```

If the plots convey different types of information, then they should be created separately and then

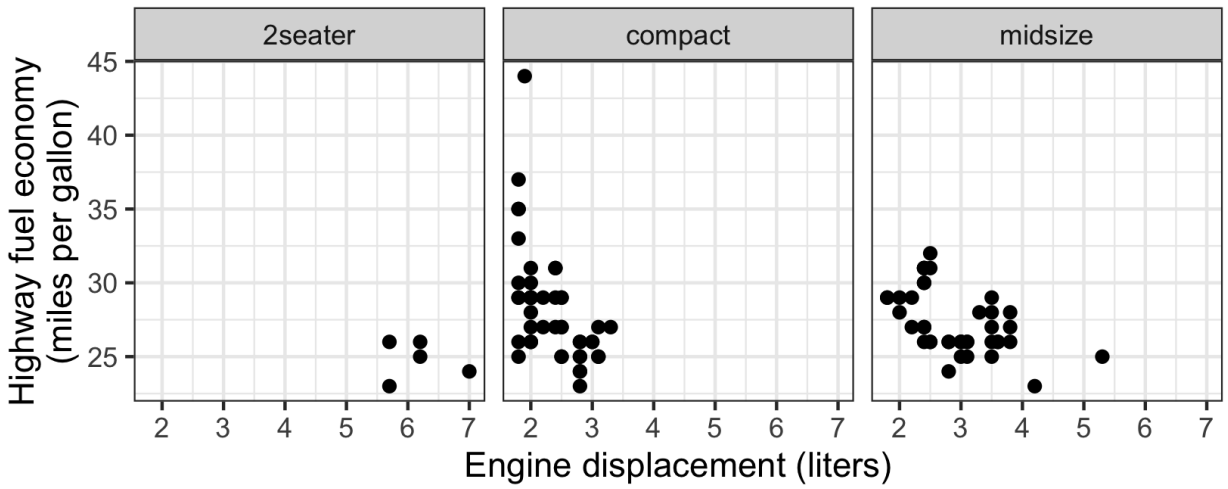


Figure 9: An illustration of using `facet_wrap` to create a multi-panel plot.

concatenated together using `cowplot::plot_grid`. An example is shown below and in Figure 10. Note that the figure caption should reference the subpanels by their labels (in this case, a and b).

```
# illustration of using cowplot to concatenate multiple plots

# first plot: box plot of fuel economy by car type
p1 = mpg %>%
  mutate(class =                          # re-order car classes by fuel economy
           fct_reorder(class, hwy)) %>%
  ggplot(aes(x = class, y = hwy, fill = class)) +
  geom_boxplot() +
  labs(
    x = "Car type",
    y = "Highway fuel economy\n(miles per gallon)"
  ) +
  theme_bw() +
  theme(legend.position = "none",         # remove legend and x axis text because
        axis.text.x = element_blank())    #  information present in second plot

# second plot: scatter plot of fuel economy versus car type
p2 = mpg %>%
  mutate(class =                          # re-order car classes by fuel economy
           fct_reorder(class, hwy)) %>%
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (liters)",
    colour = "Car type"
  ) +
```

```
  theme_bw() +
  theme(axis.title.y = element_blank())  # remove y axis title because already
                                         #  present in the first plot

# use cowplot to concatenate the two plots
p = cowplot::plot_grid(p1, p2,
                       labels = "auto",      # generate labels for subplots
                       rel_widths = c(1,2), # specify relative widths
                       align = "h")         # how to align subplots

# save the plot
ggsave(plot = p, filename = "cowplot-demo.png",
       device = "png", width = 5, height = 2.5)
```

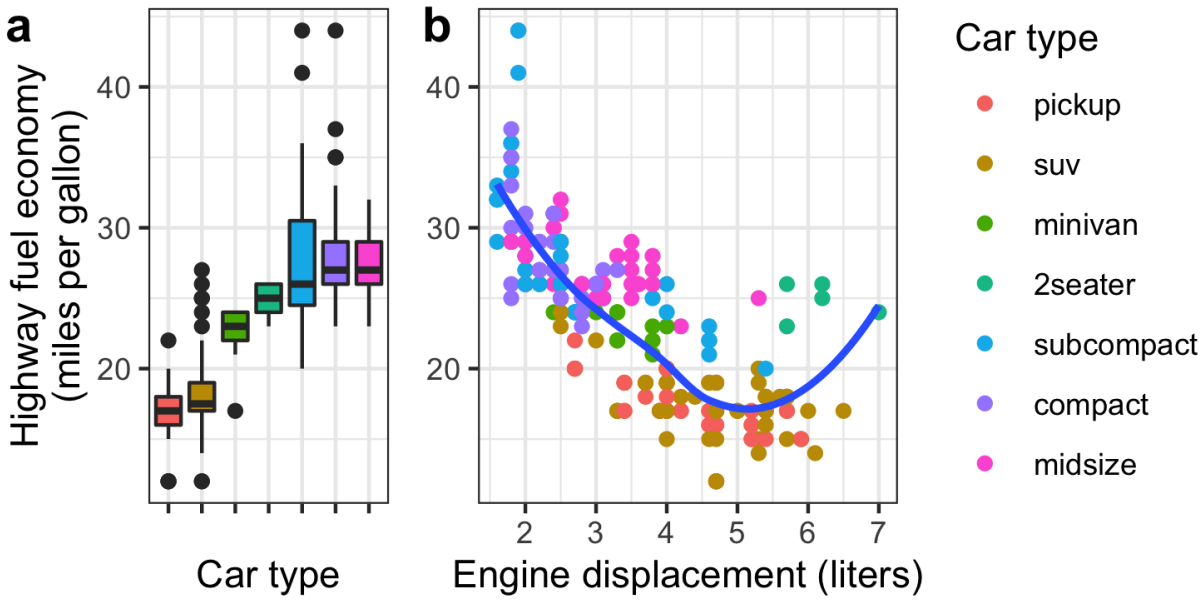


Figure 10: (An illustration of using `cowplot` to create a multi-panel plot.) Relationships between highway fuel economy and car type (a) and engine displacement (b).

3.4 Tables

Tables are generally less complex than figures, but many of the principles of creating high-quality figures carry over to tables as well (e.g. choosing appropriate sizes, captions, and titles.)