Preparing homeworks and exams in STAT 471

Eugene Katsevich

August 28, 2021

1 Compilation

Compile your R Markdown file to PDF by pressing the `Knit` 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:

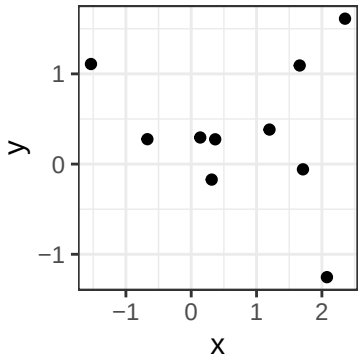
- Make sure you have followed all the steps in getting-started.pdf. This document may have been updated since you last saw it.
- Avoid using underscores or other special characters in chunk headers or figure/table captions.
- You might be missing necessary R packages. Install these using `install.packages`.
- 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.
- 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

You can add a figure by plotting it inside of a code chunk:

```
```{r, fig.align = "center"}
library(tidyverse)
test_data = tibble(x = rnorm(10), y = rnorm(10))
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
```



Each figure should have a caption and should be referenced in the text. You can add a caption by including `fig.cap` in the chunk header, and you can reference the figure by including a chunk name (in this case `test-plot`). Note that having a caption is necessary to be able to reference a figure!

```{r test-plot, fig.cap = "This is a test plot.", fig.align = "center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```

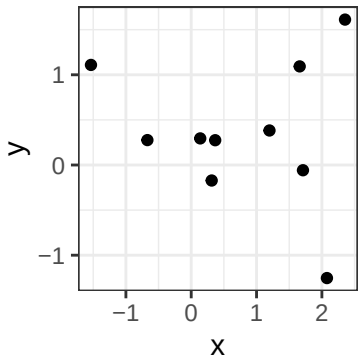


Figure 1: This is a test plot.

This produces Figure 1. This type of figure reference can be obtained by typing `Figure \@ref(fig:test-plot)`.

2.2 Tables

The simplest way to add a table is by printing it inside of a code chunk:

```{r}
test_data
```

```
## # A tibble: 10 x 2
##         x       y
##     <dbl>   <dbl>
##  1  0.138  0.295
##  2  1.66   1.09
##  3  2.08  -1.25
##  4  0.313 -0.172
##  5  1.20   0.383
##  6  2.35   1.61
##  7  1.71  -0.0582
##  8  0.367  0.275
##  9 -1.53   1.11
## 10 -0.671  0.276
```

A better way is to use the `kable` and `kable_styling` functions from the `kableExtra` package:

```{r test-table}
library(kableExtra)
test_data %>%
  kable(format = "latex", row.names = NA,
      booktabs = TRUE, digits = 2,
      caption = "This is a test table") %>%
  kable_styling(position = "center")
```

This produces Table 1. This type of table reference can be obtained by typing `Table \@ref(tab:test-table)`. Note that captions for tables must go into the `kable` function rather than into the chunk header like for

Table 1: This is a test table

x	y
0.14	0.29
1.66	1.09
2.08	-1.25
0.31	-0.17
1.20	0.38
2.35	1.61
1.71	-0.06
0.37	0.27
-1.53	1.11
-0.67	0.28

figures. The chunk name is still used to reference the table. Note that having a caption is necessary to be able to reference a table!

3 High-quality reports

Aside from data mining, another goal of STAT 471 is to teach you how to produce high-quality reports. This skill is essential to successfully communicating the results of your data analyses. Therefore, each submitted homework and exam will be held to a high standard of presentation, which will be evaluated and will comprise a 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 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_correlation =
```

```
## # A tibble: 1 x 1
##   sample_correlation
##                <dbl>
## 1              0.742
```

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               0.719
```

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.

3.3.1 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 (specified in the chunk options via the `fig.width` and `fig.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:

````
```{r test-plot-abs-small, echo = FALSE, fig.cap = "This plot's absolute
size is too small.", fig.width=1, fig.height=1, out.width="40%", fig.align = "center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
````

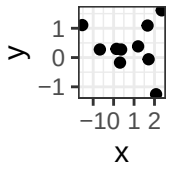


Figure 2: This plot's absolute size is too small.

````
```{r test-plot-abs-medium, echo = FALSE, fig.cap = "This plot's absolute
size is about right.", fig.width=2, fig.height=2, out.width="40%", fig.align = "center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
````

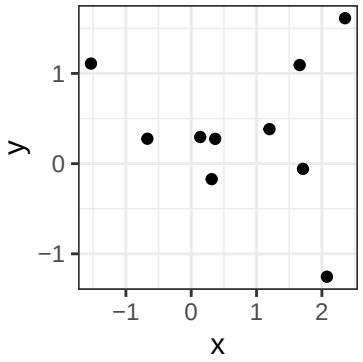


Figure 3: This plot's absolute size is about right.

````
```{r test-plot-abs-large, echo = FALSE, fig.cap = "This plot's absolute
size is too large.", fig.width=5, fig.height=5, out.width="40%", fig.align = "center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
````

Figures 2, 3, 4 are produced by these three code chunks. 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

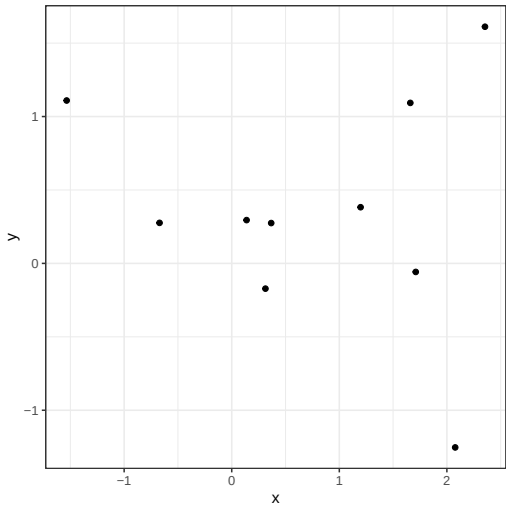


Figure 4: This plot's absolute size is too large.

rule of thumb is that the smallest text in your plots should be roughly the same size as the text in your report.

The **relative size** of your figures (relative to the dimensions of your report, as specified by `out.width` in the chunk header) should also be chosen appropriately. Compare Figures 5, 6, and 7, corresponding to relative sizes of 10%, 40%, and 80%. The small plot is too small to see, the large plot takes up too much space, and the medium one is about right.

```
```{r test-plot-rel-small, echo = FALSE, fig.cap = "This plot's relative
size is too small.", fig.width=2, fig.height=2, out.width="10%", fig.align =
"center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
```

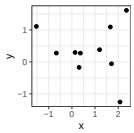


Figure 5: This plot's relative size is too small.

```
```{r test-plot-rel-medium, echo = FALSE, fig.cap = "This plot's relative
size is about right.", fig.width=2, fig.height=2, out.width="40%", fig.align =
"center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
```

```
```{r test-plot-rel-large, echo = FALSE, fig.cap = "This plot's relative
size is too large.", fig.width=2, fig.height=2, out.width="80%", fig.align =
"center"}
test_data %>% ggplot(aes(x = x, y = y)) + geom_point() + theme_bw()
```
```

3.3.2 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 8.

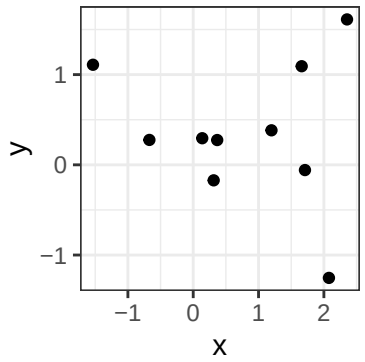


Figure 6: This plot's relative size is about right.

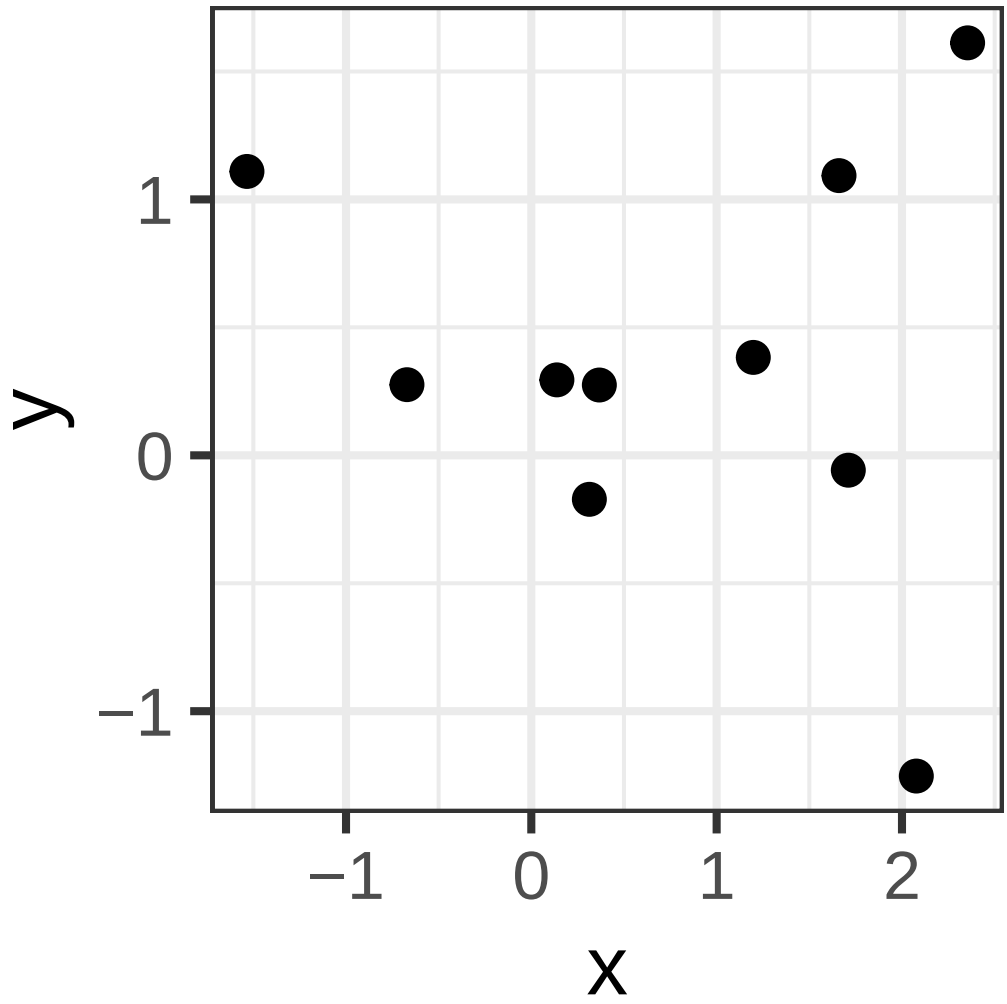


Figure 7: This plot's relative size is too large.

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

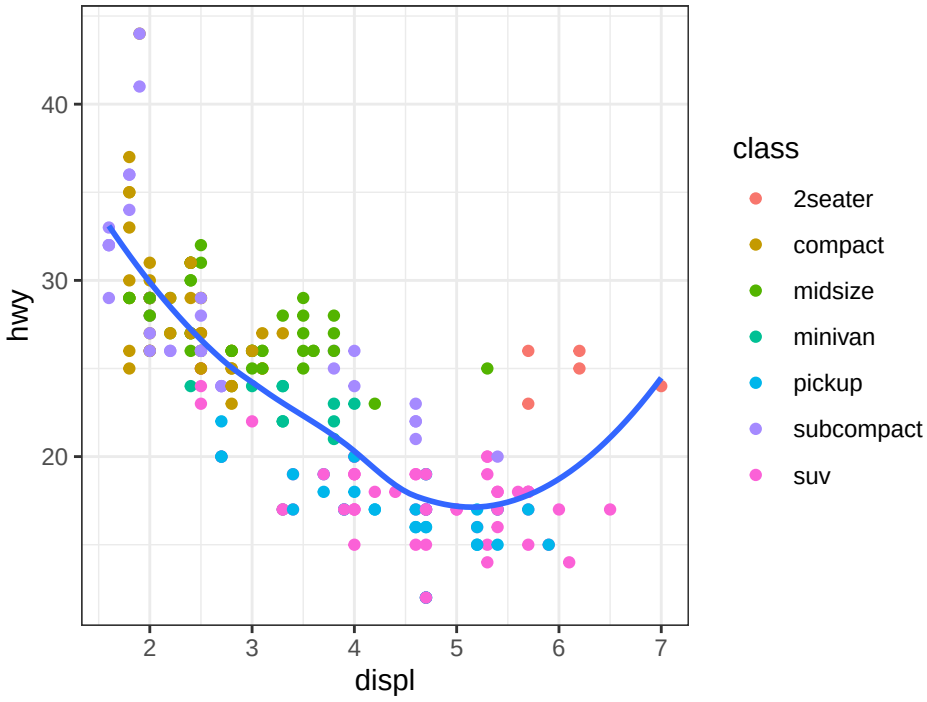


Figure 8: A plot without clear titles.

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 9, which is much easier to understand.

```
# a plot with clear axis and legend titles
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()
```

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 9.) 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 9.

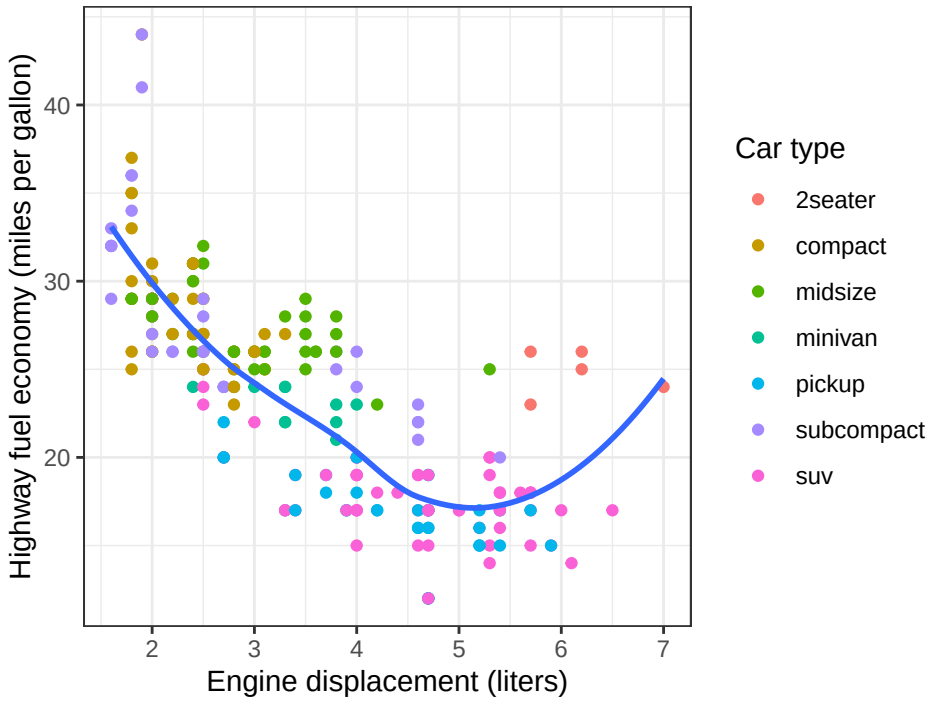


Figure 9: (A plot with clear axis and legend titles). Fuel efficiency generally decreases with engine size; two-seaters (sports cars) are an exception because of their light weight.

3.3.3 Captions

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

3.3.4 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 10 illustrates `facet_wrap` for the `mpg` data used in Figures 8 and 9.

```
# illustrate how to use facet_wrap to create a multi-panel plot
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()
```

If the plots convey different types of information, then they should be created separately and then concatenated together using the `plot_grid` function from the `cowplot` package. An example is shown below and in Figure 11.

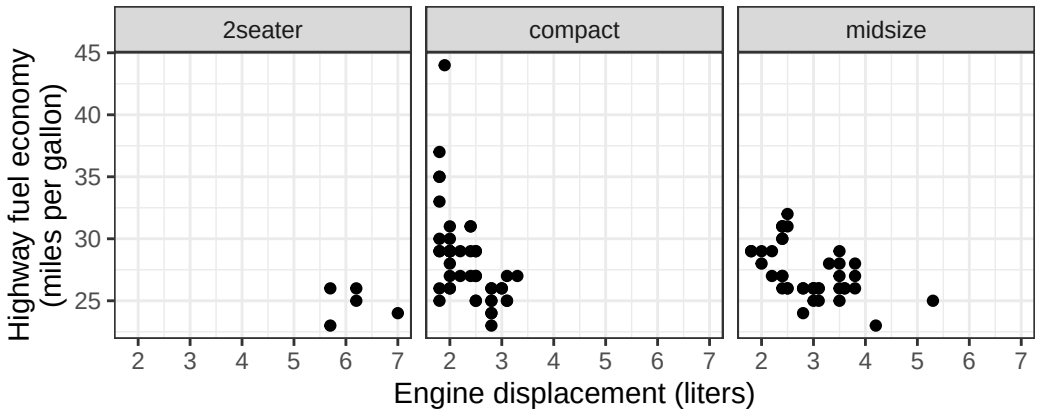


Figure 10: An illustration of using `facet_wrap` to create a multi-panel plot.

```
# 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
library(cowplot)
plot_grid(p1, p2,
          rel_widths = c(1,2), # specify relative widths
          align = "h")         # how to align subplots
```

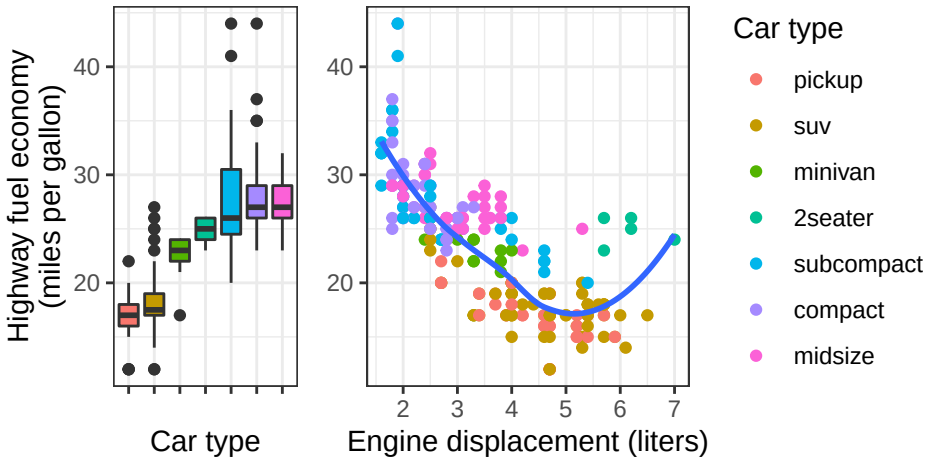


Figure 11: (An illustration of using `cowplot` to create a multi-panel plot.) Relationships between highway fuel economy and car type (left panel) and engine displacement (right panel).

Table 2: A table without clear column titles

cyl	drv	n
4	4	23
4	f	58
5	f	4
6	4	32
6	f	43
6	r	4
8	4	48
8	f	1
8	r	21

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.)

3.4.1 Column titles for tables

Just like axis labels for figures, column titles for tables should be easily readable. Often this means not using the variable names directly from your data frame. For example, consider Table 2, which tabulates the number of cylinders and the drive train type for the cars in `mpg`.

```
# a table without clear column titles
mpg %>%
  count(cyl, drv) %>%
  kable(format = "latex", row.names = NA,
      booktabs = TRUE, digits = 2,
      caption = "A table without clear column titles") %>%
  kable_styling(position = "center")
```

We can specify clear column names via the `col.names` argument to `kable`. See Table 3 and the code chunk that produced it.

Table 3: (A table with clear column titles.) Cross-tabulation of the number of cylinders and the drive train type for the cars in mpg.

Num. cylinders	Drive train	Count
4	4	23
4	f	58
5	f	4
6	4	32
6	f	43
6	r	4
8	4	48
8	f	1
8	r	21

```
# a table with clear column titles
mpg %>%
  count(cyl, drv) %>%
  kable(format = "latex",
        row.names = NA,
        col.names = c("Num. cylinders", "Drive train", "Count"),
        booktabs = TRUE,
        digits = 2,
        caption = "(A table with clear column titles.) Cross-tabulation of the
        number of cylinders and the drive train type for the cars in mpg.") %>%
  kable_styling(position = "center")
```