Preparing homeworks and exams in STAT 4710

Eugene Katsevich

August 23, 2023

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). The PDF will pop up in a separate browser tab. You can download this PDF for submission to Gradescope.

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

- Avoid using underscores or other special characters in chunk headers or figure/table captions.
- Put a blank line between each code chunk and the subsequent text, especially for code chunks containing figures.
- You might not have loaded all necessary R packages. See Section 2.
- 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.
- Try Googling your error messages and/or asking ChatGPT.
- If you are stuck, post on Ed Discussion or come to office hours and the teaching staff will assist you.

2 Loading R packages

If your code requires functions from packages not in base R, you must load these packages via `library`. For example:

```
library(tidyverse)  # for tidyverse packages
library(kableExtra) # for printing tables
library(cowplot)    # for side-by-side plots
```

Note that `library(tidyverse)` loads all the core `tidyverse` packages, including `ggplot2` for data visualization, `dplyr` for data manipulation, `tidyr` for data tidying, and `readr` for data import.

3 Adding figures and tables to your report

Before getting started, let's add global R Markdown chunk configurations as follows:

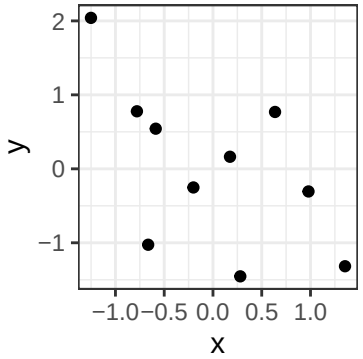
````
```{r, echo = FALSE}
knitr::opts_chunk$set(fig.pos = "!h", fig.align = "center")
ggplot2::theme_set(ggplot2::theme_bw())
```
````

This will set the horizontal and vertical alignment of figures: horizontally they will be centered, and vertically they will appear right after the code used to produce them. Furthermore, it will add a nice theme to the plots produced.

3.1 Figures

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

```
```{r}
test_data <- tibble(x = rnorm(10), y = rnorm(10))
test_data |>
  ggplot(aes(x = x, y = y)) +
  geom_point()
```
```



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."}
test_data |>
  ggplot(aes(x = x, y = y)) +
  geom_point()
```
```

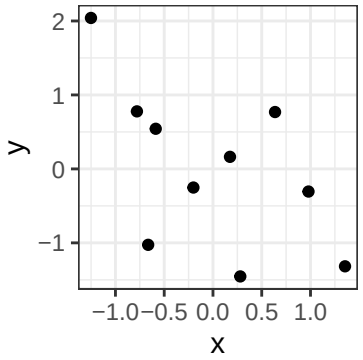


Figure 1: This is a test plot.

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

3.2 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 -0.587  0.542
##  2 -1.25   2.04
##  3  0.636  0.769
##  4 -0.780  0.778
##  5  1.35  -1.32
##  6  0.979 -0.306
##  7 -0.665 -1.03
##  8 -0.201 -0.252
##  9  0.280 -1.45
## 10  0.175  0.163
```

In R, tables can be stored as either *data frames* or *tibbles*. Tibbles are the tidyverse's upgraded version of data frames. One advantage of tibbles is that they look nicer when printed. In particular, if you have a tibble with many rows, then only the first few rows will be displayed when it is printed. On the other hand, printing a data frame with many rows will lead to all the rows being printed, which can unnecessarily add dozens of pages to your report! Please make sure your tables are in tibble form prior to printing them. Note that data read into R using tidyverse import functions (e.g. `read_csv()`) will automatically be in tibble format. On the other hand, data read into R using base R import functions (e.g. `read.csv()`) will be in data frame format.

A better way than simply printing a table is to use the `kable` and `kable_styling` functions from the `kableExtra` package:

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

Table 1: This is a test table

x	y
-0.59	0.54
-1.25	2.04
0.64	0.77
-0.78	0.78
1.35	-1.32
0.98	-0.31
-0.66	-1.03
-0.20	-0.25
0.28	-1.45
0.17	0.16

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 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! The code `latex_options = "HOLD_position"` ensures that the table appears right after the code used to create it.

4 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.

4.1 Text

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

4.2 Code

Your code should be commented and easy to read. The level of commenting in this document is a good guideline. Each contiguous chunk of code has a comment at the top, and then some individual lines that deserve extra attention/clarification have comments as well. There is no need to comment every single line of code. When in doubt, however, feel free to add a comment; we will not deduct points for "too much" commenting.

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              0.753
```

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              0.781
```

Your code should conform to the style guidelines described in Chapter 5 of R for Data Science. You can easily style your code with the help of the `styler` package (installed for you on Posit Cloud) by clicking `Addins -> Style selection` or `Addins -> Style active file`.

4.3 Figures

Figures are very important tools to convey information to readers, and they should be constructed thoughtfully. Please read Chapter 12 of R for Data Science, which is a good reference for producing high-quality figures. Here we discuss some of the most important elements.

4.3.1 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 **size** of your figures (specified in the chunk options via the `fig.width` and `fig.height` arguments, which are in inches) should be such that the the text on the plot is easy to read. Consider the following three choices for the size of the test plot from Figure 1:

```
```{r test-plot-abs-small, echo = FALSE, fig.cap = "This plot's
size (1in by 1in) is too small.", fig.width=1, fig.height=1}
test_data |>
  ggplot(aes(x = x, y = y)) +
  geom_point()
```
```

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

```
```{r test-plot-abs-large, echo = FALSE, fig.cap = "This plot's
size (5in by 5in)  is too large.", fig.width=5, fig.height=5,}
test_data |>
  ggplot(aes(x = x, y = y)) +
  geom_point()
```
```

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 rule of thumb is that the smallest text in your plots should be roughly the same size as the text in your report. Some experimentation is necessary to find an appropriate size for each plot.

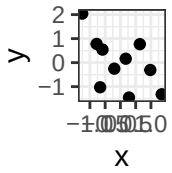


Figure 2: This plot's size (1in by 1in) is too small.

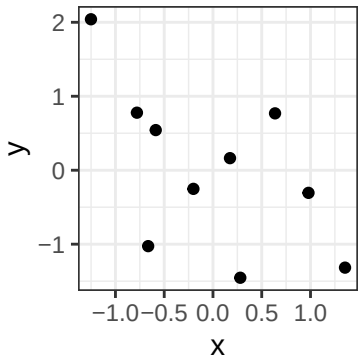


Figure 3: This plot's size (2in by 2in) is about right.

4.3.2 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 5.

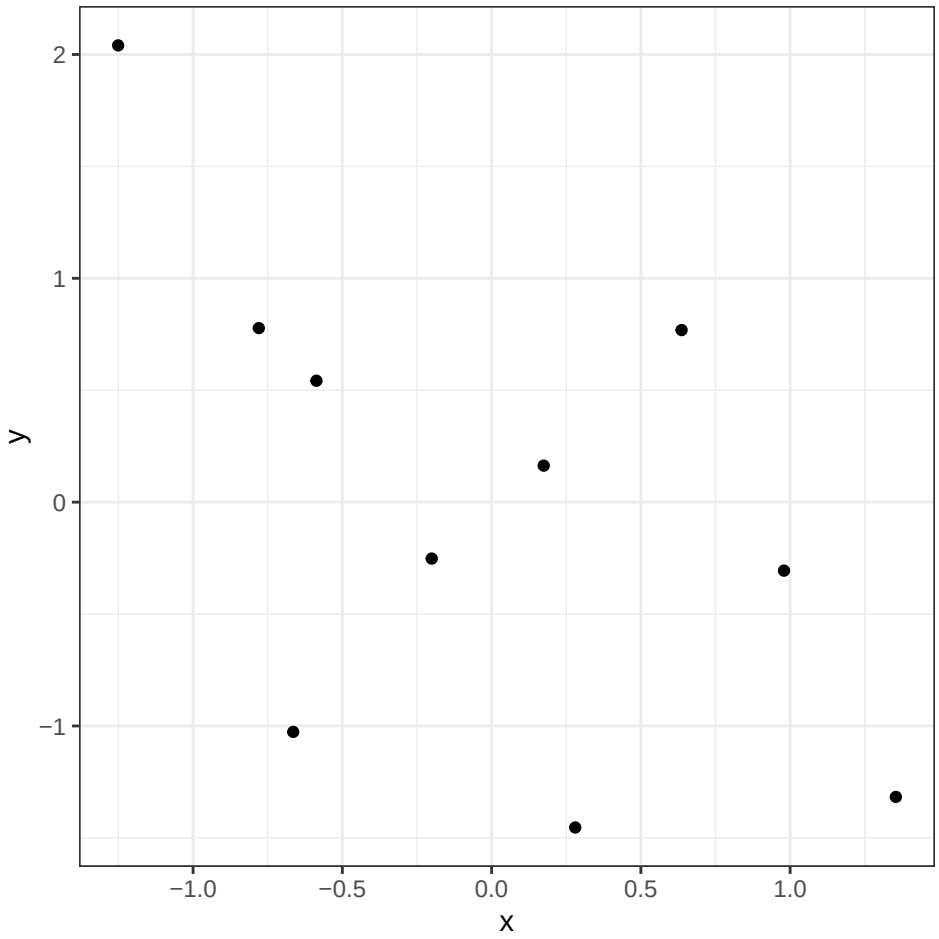


Figure 4: This plot's size (5in by 5in) is too large.

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

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 6, 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"
  )
```

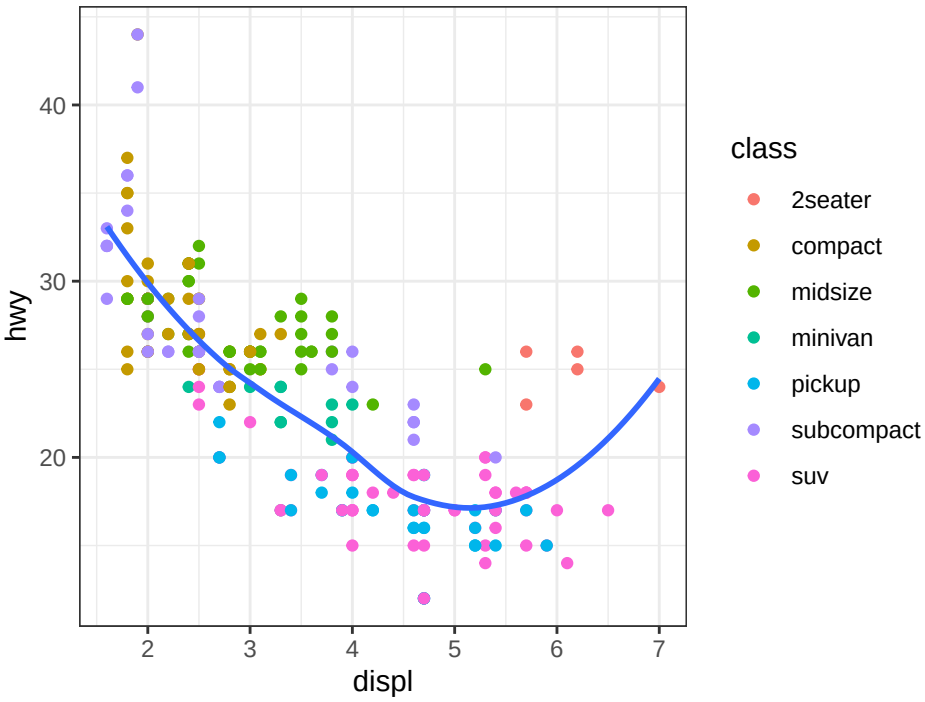


Figure 5: A plot without clear titles.

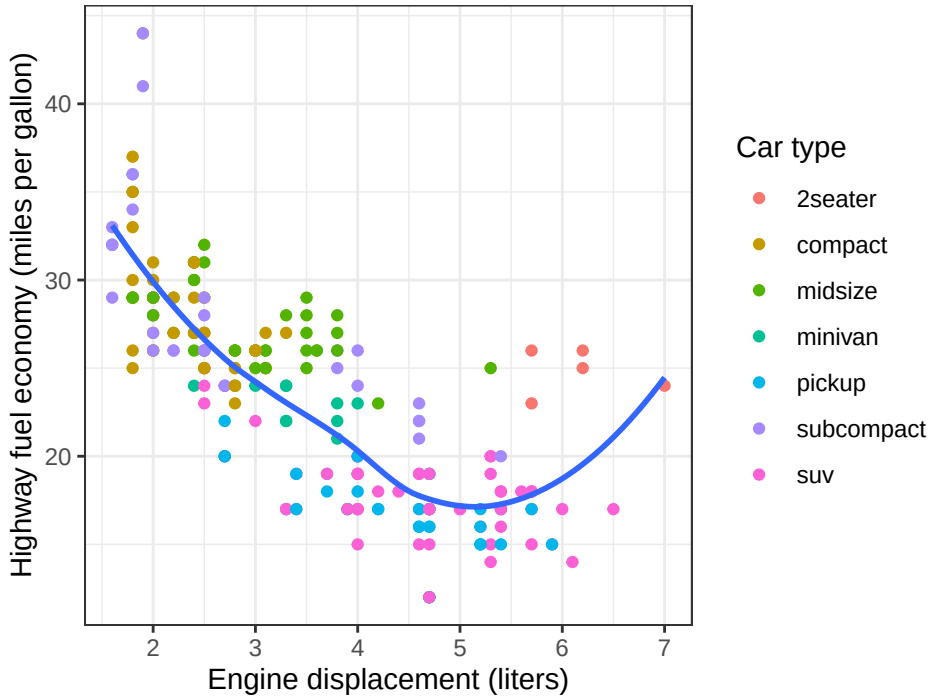


Figure 6: (A plot with clear axis and legend titles). Fuel efficiency generally decreases with engine size; two-seaters (sports cars) are an exception because of their light weight.

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 6.) To add plot titles if necessary, use the `title` argument of `labs()`. If applicable, axis titles should also include the units of measurement, e.g. liters or miles per

gallon as in Figure 6.

4.3.3 Captions

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

4.3.4 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 7 illustrates `facet_wrap` for the `mpg` data used in Figures 5 and 6.

```
# 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
  )
```

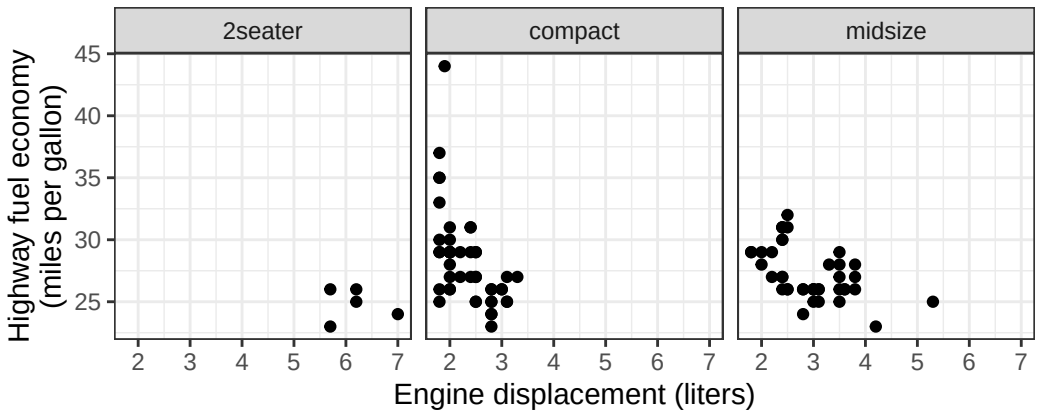


Figure 7: An illustration of using `facet_wrap` to create a multi-panel plot.

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 8.

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

# first plot: box plot of fuel economy by car type
p1 <- mpg |>
  mutate(
    class = fct_reorder(class, hwy) # re-order car classes by fuel economy
  ) |>
  ggplot(aes(x = class, y = hwy, fill = class)) +
  geom_boxplot() +
  labs(
```

```
    x = "Car type",
    y = "Highway fuel economy\n(miles per gallon)"
  ) +
  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 = fct_reorder(class, hwy) # re-order car classes by fuel economy
  ) |>
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (liters)",
    colour = "Car type"
  ) +
  theme(axis.title.y = element_blank()) # remove y axis title because already
                                        #  present in the first plot

# use cowplot to concatenate the two plots
plot_grid(p1, p2,
  rel_widths = c(1, 2), # specify relative widths
  align = "h"           # how to align subplots
)
```

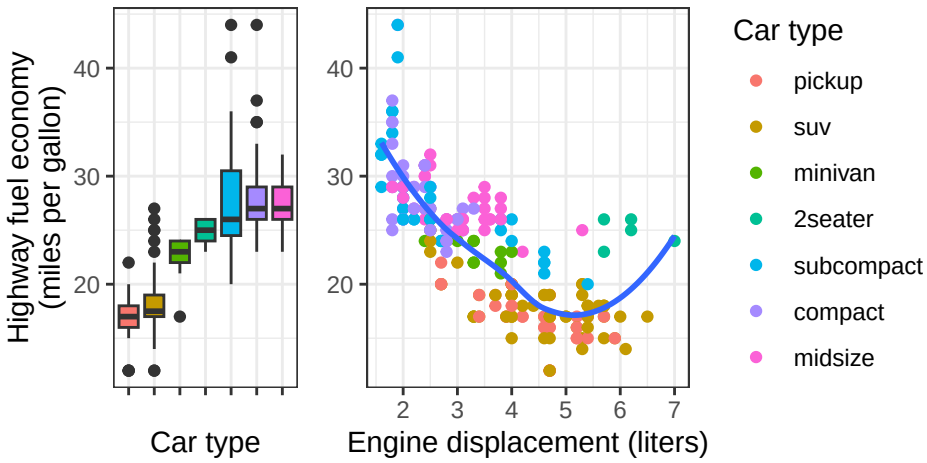


Figure 8: (An illustration of using `cowplot` to create a multi-panel plot.) Relationships between highway fuel economy and car type (left panel) and engine displacement (right panel).

4.4 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.)

4.4.1 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", latex_options = "HOLD_position")
```

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

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

```
# 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", latex_options = "HOLD_position")
```

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