Chapter 2

Data Visualization with the ggplot2 Package

EXAM PA LEARNING OBJECTIVES

3. Topic: Data Visualization

Learning Objectives

The Candidate will be able to create effective graphs in RStudio.

Learning Outcomes

a) Understand the key principles of constructing graphs.

b) Create a variety of graphs using the ggplot2 package.

4. Topic: Data Types and Exploration

e) Apply univariate and bivariate data exploration techniques.

Chapter overview: An integral part of any predictive analytic exercise is the use of graphical displays to visualize the relationships between variables of interest. In this regard, one of the key strengths of R as a programming language is that it offers versatile graphing capabilities, both in the base installation and with add-on packages. A wide variety of high-quality graphs can be produced with a minimal amount of code. In Exam PA, you are asked to take advantage of R's graphing capabilities and supplement your written report with the aid of different types of graphical displays. Instead of using R's base graphical platform, Exam PA specifically requires that you produce graphs using the `ggplot2` package,[i] which is probably new to you even if you have used R before. Compared to R's base graphics system, `ggplot2` calls for a vastly different syntax based on the so-called "grammar of graphics" (in fact, "gg" stands for "grammar of graphics") and lends its way to producing sophisticated graphs that would be cumbersome to create using base R graphics.

Synthesizing the material in the first four chapters of the book *Data Visualization: A Practical Introduction*, this chapter presents some of the most important graphical functions in the `ggplot2` package for Exam PA. These functions can be used to construct a wide variety of graphs such as

[i]The `ggplot2` package is developed by Hadley Wickham, who is also a core developer of RStudio. Earlier the package was called `ggplot`, but later substantial changes were made, so the name of the package was upgraded to `ggplot2`.

scatterplots, histograms, and bar charts. In Section 2.1, we will learn the basic structure of a ggplot, make some simple but informative plots, and learn how to tweak the appearance of a ggplot, all in the context of a small-scale dataset of automobiles. Section 2.2 draws upon the data visualization techniques covered in Section 2.1 to perform data exploration. Using a real insurance dataset with more than 20,000 observations, we introduce additional geoms and illustrate the use of ggplots to uncover patterns and relationships and generate hypotheses which can be answered in a predictive model.

2.1 Fundamentals of a ggplot

2.1.1 Basic Features

Let's begin by installing and loading the `ggplot2` package.

```
# CHUNK 1
#install.packages("ggplot2") # uncomment this line the first time you use ggplot2
library(ggplot2)
```

Skeleton. In `ggplot2`, a plot consists of two parts: The core `ggplot()` function (not `ggplot2()`!) and a chain of additional functions that define the type of plot pasted using the plus (+) sign.

1. `ggplot()` *function:* The `ggplot()` function initializes the plot, defines the source of data using the `data` argument (almost always a data frame in Exam PA; recall Subsection 1.2.3), and, most importantly, specifies what variables in the data are "mapped" to visual elements in the plot by the `mapping` argument. Mappings in a ggplot are specified using the `aes()` function, with `aes` standing for "aesthetics." They determine the role different variables play in the plot. The variables may, for instance, correspond to visual elements such as the x- or y-variables, color, size, and shape, specified by the `x`, `y`, `color`, `size`, and `shape` aesthetics, respectively.

2. *Geom functions:* Subsequent to the `ggplot()` function, we put in *geometric objects*, or *geoms* for short, which include points, lines, bars, histograms, box plots, and many other possibilities, by means of one or more *geom functions*. Placed layer by layer, these geoms determine what kind of plot is to be drawn and modify its visual characteristics, taking the data and aesthetic mappings specified in the `ggplot()` function as inputs.

Here is the generic structure of a ggplot:

```
ggplot(data = <DATA>, mapping = aes(<AESTHETIC_1> = <VARIABLE_1>,
                                    <AESTHETIC_2> = <VARIABLE_2>,
                                    ...)) +
  geom_<TYPE>(<...>) +
  geom_<TYPE>(<...>) +
  <OTHER_FUNCTIONS>  +
  ...
```

```
# CHUNK 2
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point()
```

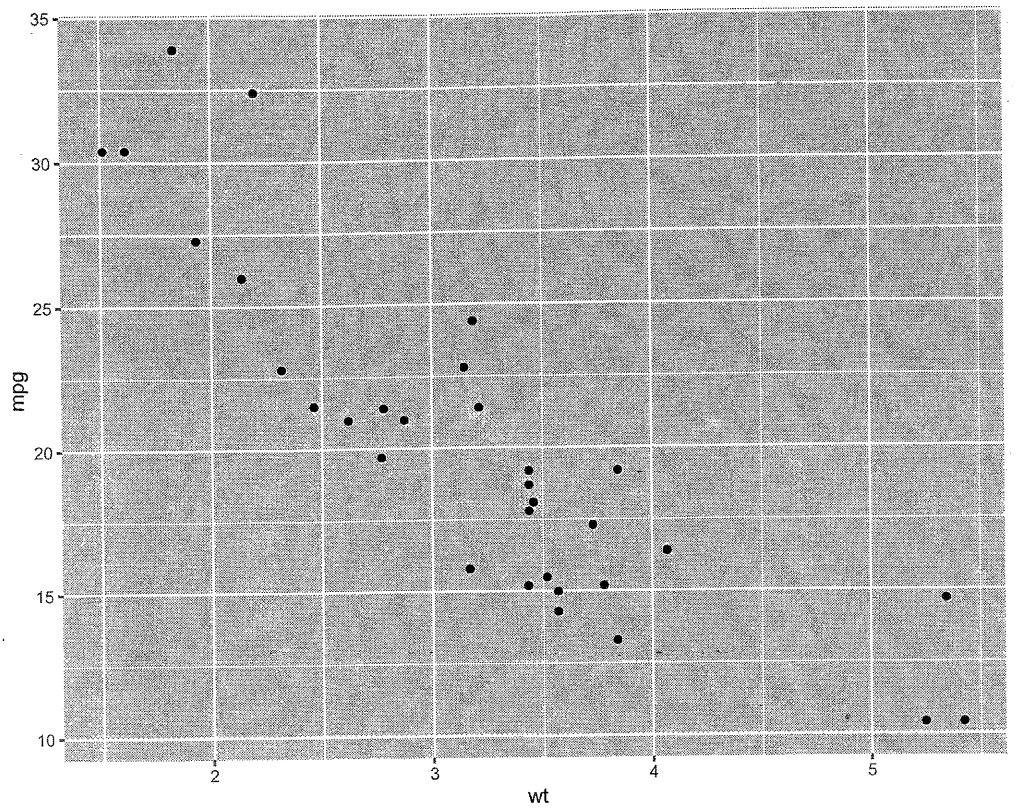

Figure 2.1.1: A scatterplot of miles per gallon against weight in the `mtcars` dataset.

Don't worry if the ideas above seem puzzling—it is commonly acknowledged that the learning curve of ggplot2 is steep, much more so than R's base graphics system. You will gain a much better understanding of how a ggplot works after going through the example plots in this chapter and in the rest of this study manual.

First encounter with ggplot. As our first example, we consider a very simple ggplot based on the `mtcars` dataset that comes with R's base installation (type `?mtcars` to learn more about the dataset). The resulting plot, run by the code in CHUNK 2, is given in Figure 2.1.1. The first line of the code makes it clear that we are using the `mtcars` data, where the variables `wt` (weight) and `mpg` (miles per gallon) are mapped to the variables on the x-axis and y-axis, respectively. There is no need to name the variables as `mtcars$wt` and `mtcars$mpg` as the data source is already specified in the `data` argument.[ii] Given these mappings, we use `geom_point()` to make a scatterplot of `mpg` (the y-variable) against `wt` (the x-variable), which allows us to see the two variables in comparison with each other. Later, we will fine-tune this scatterplot in different ways to bring out different sorts of information.

[ii] The code still works if you change `wt` and `mpg` to `mtcars$wt` and `mtcars$mpg`, respectively, even though the long names are unnecessary.

```
# CHUNK 3
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, color = "blue")) +
  geom_point()
```

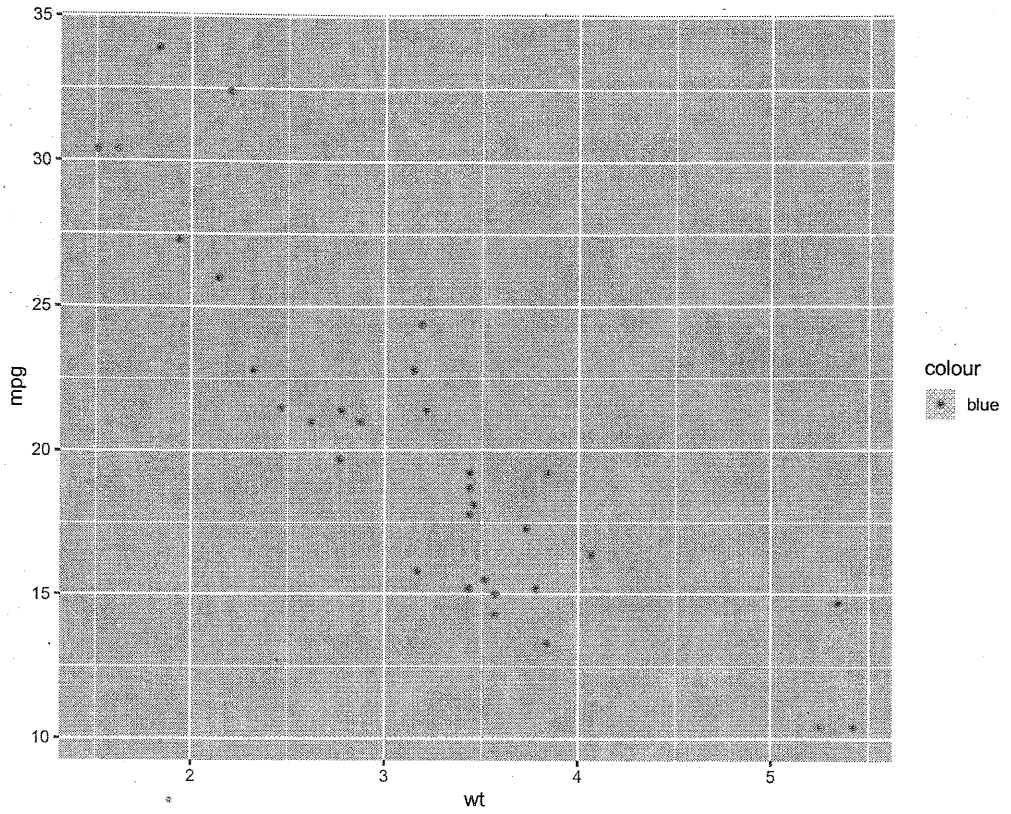

Figure 2.1.2: A version of Figure 2.1.1 with all of the points inadvertently colored in red.

Using aesthetics the right way: The essence of aesthetic mappings. One of the most common ways to modify the appearance of a plot is to color the observations in order to produce a more impressive visual effect. To color the data points say, in blue, you may be tempted to make use of the `color`[iii] aesthetic and simply insert `color = "blue"` as an additional argument to the `aes()` function as in CHUNK 3. Doing so will produce unexpected and undesirable results as shown in Figure 2.1.2. To your astonishment, all of the data points are colored in red instead of blue and there is a legend saying "blue." What has gone awry here?

Bear in mind that an aesthetic is a mapping between variables in our data and visual properties of the graph. The use of `color = "blue"` instructs the `aes()` function to map the `color` aesthetic to a variable named `"blue"` in the `mtcars` dataset. There is no such variable in our data, but the `aes()` function will do its best by treating `"blue"` as if it were a variable. The effect is the creation of a new character variable taking one and only one value, `"blue"`. As all observations share the same `"blue"` value, all of them will be mapped to the same color. In `ggplot2`, the default first-category hue is red (not blue!). This explains why every point in the scatterplot becomes red in color.

To do the coloring the right way, bear in mind that making all the points blue in color does

[iii] Both the American spelling (color) and British spelling (colour) are accepted.

```
# CHUNK 4
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point(color = "blue")
```

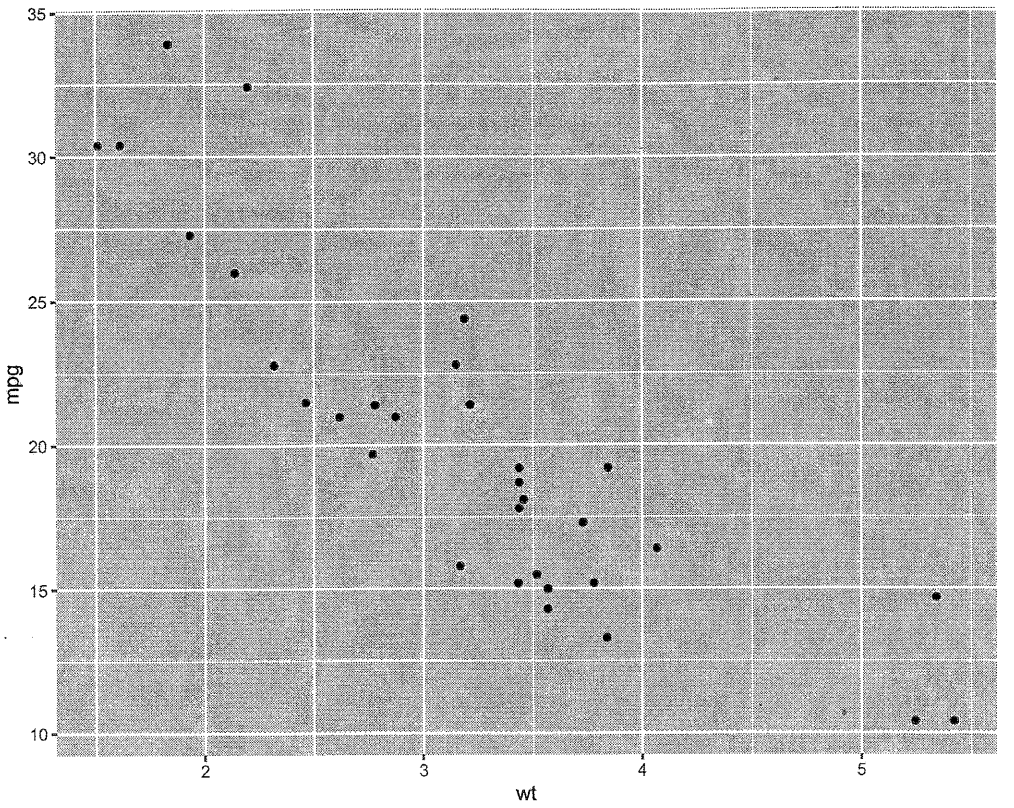

Figure 2.1.3: A version of Figure 2.1.1 with all of the points correctly colored in blue.

not involve any mapping between variables in our data and the `color` aesthetic. After all, all the observations are colored in blue and are not distinguished on the basis of color. As a result, we should not put `color = "blue"` inside the `aes()` function. It should instead be placed inside the `geom_point()` function to modify the color of the plotted points. Figure 2.1.3 shows the desired scatterplot using the code in CHUNK 4. To our liking, all of the points are colored in blue.

Figures 2.1.2 and 2.1.3 show what may go wrong if we fail to grasp the important distinction between setting an aesthetic to a constant value and mapping it to a variable. They highlight a subtle but extremely important mentality when making ggplots: The `aes()` function is reserved for mappings. To set a property that affects how a plot looks but does not involve mapping variables to aesthetic elements, we should do it outside the `aes()` function—in the geom functions. To put it another way, the aesthetics determine *what* relationships we want to see in the plot whereas the geoms determine *how* we want to see the relationships.

Now let's see an example of using the `color` aesthetic the right way. In the `mtcars` data, the `am` variable is a binary numeric variable equal to 0 for automatic cars and 1 for manual cars.

```
class(mtcars$am)

## [1] "numeric"
```

```
# CHUNK 5
mtcars$am.fac <- factor(mtcars$am, levels = c(0, 1),
                        labels = c("Automatic", "Manual"))
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, color = am.fac)) +
  geom_point()
```

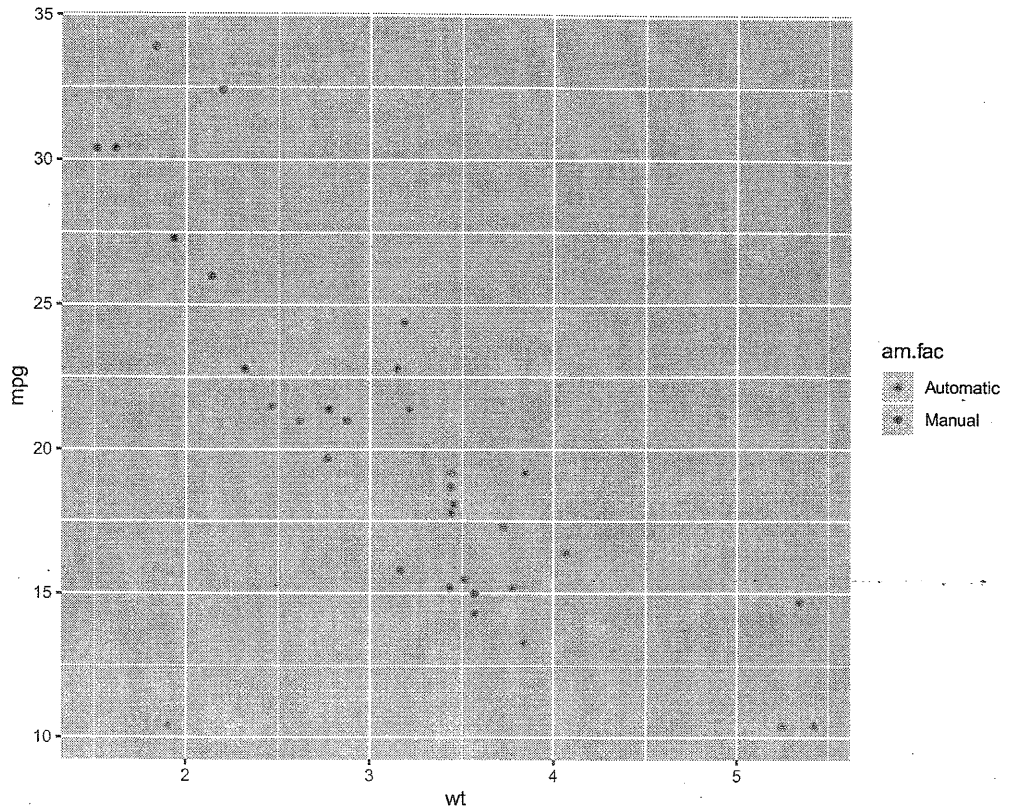

Figure 2.1.4: A version of Figure 2.1.1 with automatic cars shown in red and manual cars shown in green.

To color the different cars according to whether they are automatic or manual, we first transform the `am` variable into a two-level factor named `am.fac` via the `factor()` function (recall that factors are discussed on page 19), then map the `color` aesthetic to `am.fac`. The resulting scatterplot, generated by the code in CHUNK 5, is given in Figure 2.1.4, where automatic cars are displayed in red whereas manual cars are displayed in green. A legend is produced accordingly. Notice that there is a genuine mapping between the `am.fac` variable and `color`, with `am.fac = 0` mapped to the red color and `am.fac = 1` mapped to the green color. In other words, the observations are *grouped* on the basis of the `am.fac` variable by color. (The `color` aesthetic does not say what colors are used to discriminate cars on the basis of `am.fac`, though.)

Example 2.1.1. (Why convert `am` to `am.fac`?) To see why the conversion of `am` to the factor variable `am.fac` is needed, run the following code in CHUNK 6:

```
# CHUNK 6
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, color = am)) +
  geom_point()
```

What do you notice? Why is the coloring done in the way you observe?

Solution. When running the code (try!), a color gradient from 0 to 1 is produced. This is because the `am` variable in the `mtcars` dataset is treated by design as a continuous numeric variable even though the two levels, 0 and 1, are merely class labels that do not convey any sense of magnitude. R implicitly allows for values between 0 and 1 for the `am` variable and therefore uses the color gradient to differentiate the observations by color (though you can observe that there are only two colors in the plot, corresponding to the two extremes in the color gradient). This explains the point made in Subsection 1.2.1 that whether a categorical variable is treated as a factor affects how the resulting graphical output looks. □

Some important geoms for Exam PA. Besides `geom_point()`, there are a number of common geoms which you need to be familiar with for Exam PA. They are listed in Table 2.1 along with their commonly used arguments which affect how the plot looks.

Geom	Type of Plot	Frequently Used Arguments
`geom_bar()`	Bar chart	`fill, alpha`
`geom_boxplot()`	Box plot	`fill, alpha`
`geom_histogram()`	Histogram	`fill, alpha, bins`
`geom_point()`	Scatterplot	`color, alpha, shape, size`
`geom_smooth()`	Smoothed line	`color, fill, method, se`

Table 2.1: Some commonly used geoms in Exam PA.

The names of these geoms are pretty self-explanatory. For example, `geom_smooth()`, as its name suggests, determines a smoothed line and, by default, produces a ribbon around the line showing the standard error. It is one of the most commonly used geoms in Exam PA. The smoothed line can be generated by different statistical methods, such as a linear fit (`method = "lm"`). The default is the use of a generalized additive model (`method = "gam"`), which is beyond the syllabus of Exam PA. To switch off the standard error bands, you can set `se = FALSE`.

Many of the geoms in Table 2.1 have an `alpha` argument which controls the transparency of the plotted object on a scale from 0 (fully transparent) to 1 (opaque); the default value is 1. The transparency of the object increases by decreasing `alpha`; the lower the value of `alpha`, the more transparent the points. In the limit when `alpha` is exactly zero, the points become completely invisible. The `alpha` argument is particularly useful when there is a lot of overlapping among the data points. By setting alpha to an intermediate value, we make it easy to see where most of the observations cluster.

In Figure 2.1.5, we plot the 32 observations in the `mtcars` dataset using large points (`size = 5`) and a small amount of transparency (`alpha = 0.4`), classify them into automatic and manual cars, and fit a separate line to each kind of cars. The commands are collected in CHUNK 7. Note that:

- For each of the two types of cars, the smoothed line and the standard error ribbon are indicated by the same color (red for automatic cars and green for manual cars), which is appealing from an aesthetic perspective. The consistent coloring is achieved by mapping both the `color` aesthetic and `fill` aesthetic (which controls filled areas of bars, polygons and, in this case, the interior of standard error bands) to the `am.fac` variable.
- If you omit `fill = am.fac` (try this in R!), then the two lines will still be colored according to the type of cars due to the `color` aesthetic, but without the `fill` aesthetic, the `geom_smooth()` function will shade the standard error ribbon by its default color, which is gray.
- If you put `geom_smooth()` first, followed by `geom_point()`, then the smoothed line will be partially obscured by some of the data points, which may not be desirable.

```
# CHUNK 7
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg,
                                    color = am.fac, fill = am.fac)) +
  geom_point(size = 5, alpha = 0.4) +
  geom_smooth(method = "lm", alpha = 0.2)
```

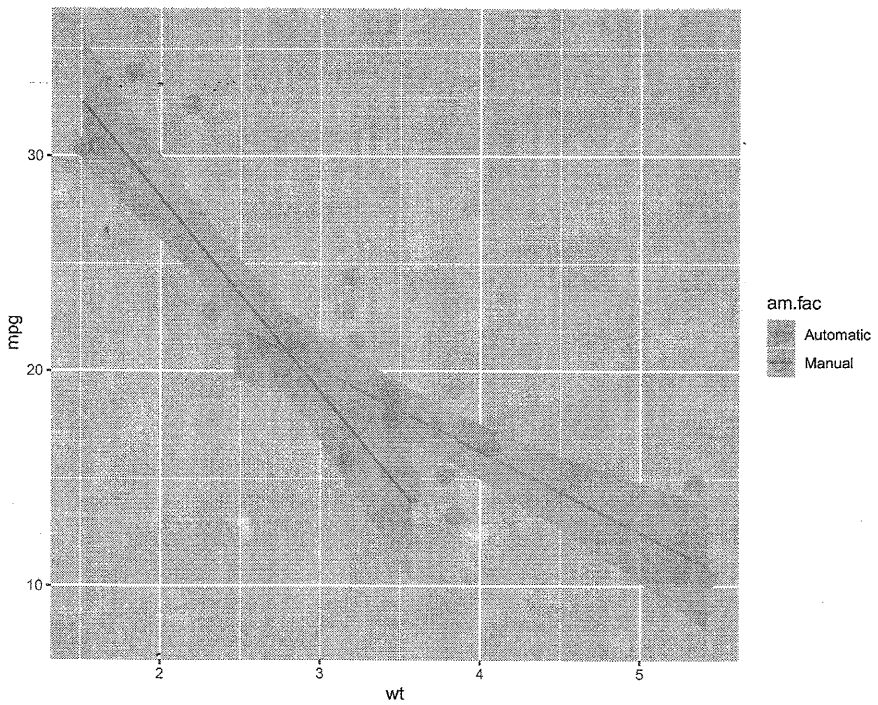

Figure 2.1.5: A version of Figure 2.1.4 with the data points enlarged and the standard error bands added.

Thanks to Figure 2.1.5, we can see that the linear relationship between `mpg` and `wt` is markedly different for the two types of car, with the line for manual cars being much steeper. In Chapter 3, we will quantify the difference between the two lines using formal statistical language. Figure 2.1.5, produced by `ggplot2`, allows us to discover such a finding in the first place and is a very useful starting point of such an investigation.

```
# CHUNK 8
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point(aes(color = am.fac), size = 5, alpha = 0.4) +
  geom_smooth(method = "lm")
```

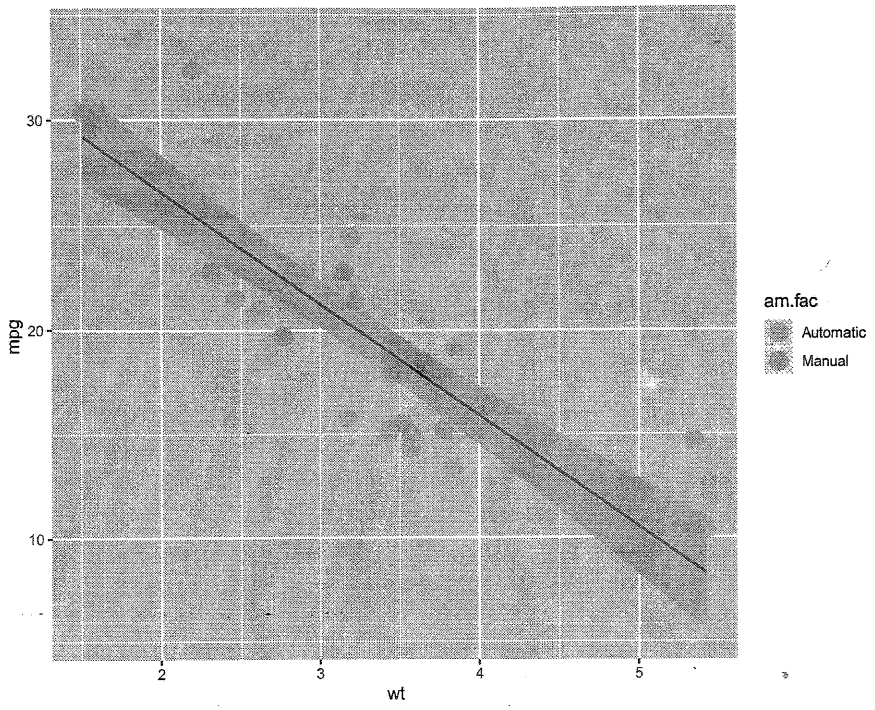

Figure 2.1.6: A scatterplot of miles per gallon against weight in the `mtcars` dataset, with a single smoothed line produced and the data points grouped by `am.fac` in color.

We will illustrate the use of `geom_bar()`, `geom_boxplot()`, and `geom_histogram()` in Section 2.2.

Geom-specific aesthetics. What if you want to make just one smoothed line applied to all 32 observations in the `mtcars` dataset while still having them colored by the type of cars? We can do so by specifying different aesthetic mappings for different geoms as in CHUNK 8. Notice that the `aes()` function in the `ggplot()` call has only the `x` and `y` aesthetics; the `color` aesthetic is moved to the `geom_point()` function. As a result, the 32 points will have their color distinguished by the `am.fac` variable. However, since there is no such mapping in the `geom_smooth()` function, a single smoothed line (colored in blue by default) fitted to all of the 32 observations surrounded by two standard error bands (colored in gray by default) will be produced as shown in Figure 2.1.6. In general, aesthetic mappings common to most, if not all, geoms can be specified in the initial `ggplot()` call. These mappings will be inherited by all geoms. If needed, you can then put in additional aesthetics that apply only to a particular geom to override the default aesthetics.

Example 2.1.2. (Variations of CHUNK 8) Consider the following variations of CHUNK 8. Mentally think about what kind of plots will be produced. Then run the code and see what happens.

```
# CHUNK 9
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, fill = am.fac)) +
  geom_point(aes(color = am.fac), size = 5, alpha = 0.4) +
  geom_smooth(method = "lm", se = FALSE)
```

```
# CHUNK 10
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point(aes(color = am.fac), size = 5, alpha = 0.4) +
  geom_smooth(aes(fill = am.fac), method = "lm")
```

Solution. Let's look at the two chunks separately.

- *CHUNK 9:* Compared to CHUNK 8, the code in CHUNK 9 has the `fill` aesthetic added to the initial `ggplot()` call and the option `se = FALSE` added to the `geom_smooth()` function. The `fill` aesthetic has no effect on the `geom_point()` function, but it does affect the `geom_smooth()` function—by default, `geom_smooth()` produces the standard error bands that are filled in color according to the `fill` aesthetic. Even though these bands are switched off due to the option `SE = FALSE`, separate smoothing lines are still fitted to the two groups of cars. Without the `color` aesthetic, however, the two lines share the same color, which is blue.
- *CHUNK 10:* Compared to CHUNK 8, the code in CHUNK 10 has the `fill` aesthetic added to the `geom_smooth()` function. As a result, separate smoothing lines are fitted to the two groups of cars with the standard error bands filled in color according to `am.fac`. As the `color` aesthetic is absent in `geom_smooth()`, the two smoothing lines still share the same color.

□

Remark. This example once again illustrates the subtlety of ggplots. A seemingly minor change in your code can lead to substantially different output.

Faceting. In Figures 2.1.4 to 2.1.6, we attempted to group the 32 observations according to the type of car (automatic vs. manual). In `ggplot2`, grouping is achieved by mapping one or more categorical variables in the data to visual elements like `color`, `shape`, `fill`, `size`, and `linetype`, as we did earlier.

We now consider *faceting*, which is another useful way to present our data. While grouping showcases two or more groups of observations in a single plot, faceting displays the observations in *separate* plots (known as a "small multiple" plot) produced for each combination of the faceting variables placed side-by-side on the same scale to facilitate comparison. In `ggplot2`, faceting is accomplished by the `facet_wrap()` function or the `facet_grid()` function, depending on how many faceting variables there are. The `facet_wrap()` function is often used when there is only one faceting variable. Its syntax is

```
# CHUNK 11
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point(size = 5, alpha = 0.4) +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ am.fac)
```

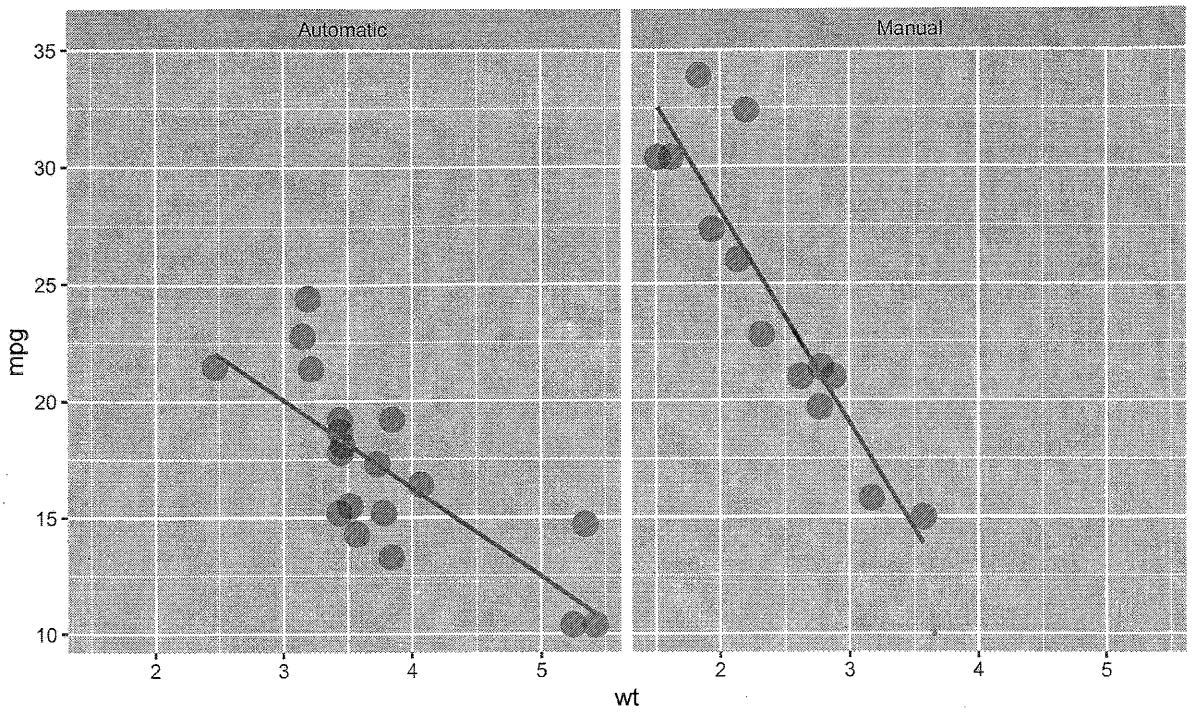

Figure 2.1.7: Faceted scatterplots of miles per gallon against weight by transmission in the `mtcars` dataset.

```
facet_wrap(~ <FACET_VAR>, ncol = <n>),
```

where the first argument specifies, following the tilde sign, the faceting variable by means of R's formula syntax (more details on formulas in R will be given in Chapter 3). The second argument, which is optional, determines the number of columns used to display the facets. The code in CHUNK 11, for example, produces the faceted scatterplot in Figure 2.1.7. The two scatterplots are laid out in order according to the levels of `am.fac`. A label is displayed at the top of each facet ("Automatic" and "Manual") for easy identification. Figure 2.1.7 is once again a manifestation of the different linear relationships exhibited by automatic and manual cars.

If there are two faceting variables, then we can do a cross-classification using the `facet_grid()` function, which produces a two-dimensional "grid" of plots. Its syntax is similar to that of `facet_wrap()`:

```
facet_grid(<FACET_VAR_1> ~ <FACET_VAR_2>, ncol = <n>),
```

Figure 2.1.8 is produced by the code in CHUNK 12 using `am.fac` and the factor version of `vs` as faceting variables. There are a total of $2 \times 2 = 4$ panels in Figure 2.1.8 as each of `am.fac` and `vs.fac` is a two-level factor.

```
# CHUNK 12
mtcars$vs.fac <- factor(mtcars$vs, levels = c(0, 1),
                        labels = c("V-shaped", "straight"))
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point(size = 5, alpha = 0.4) +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(vs.fac ~ am.fac)
```

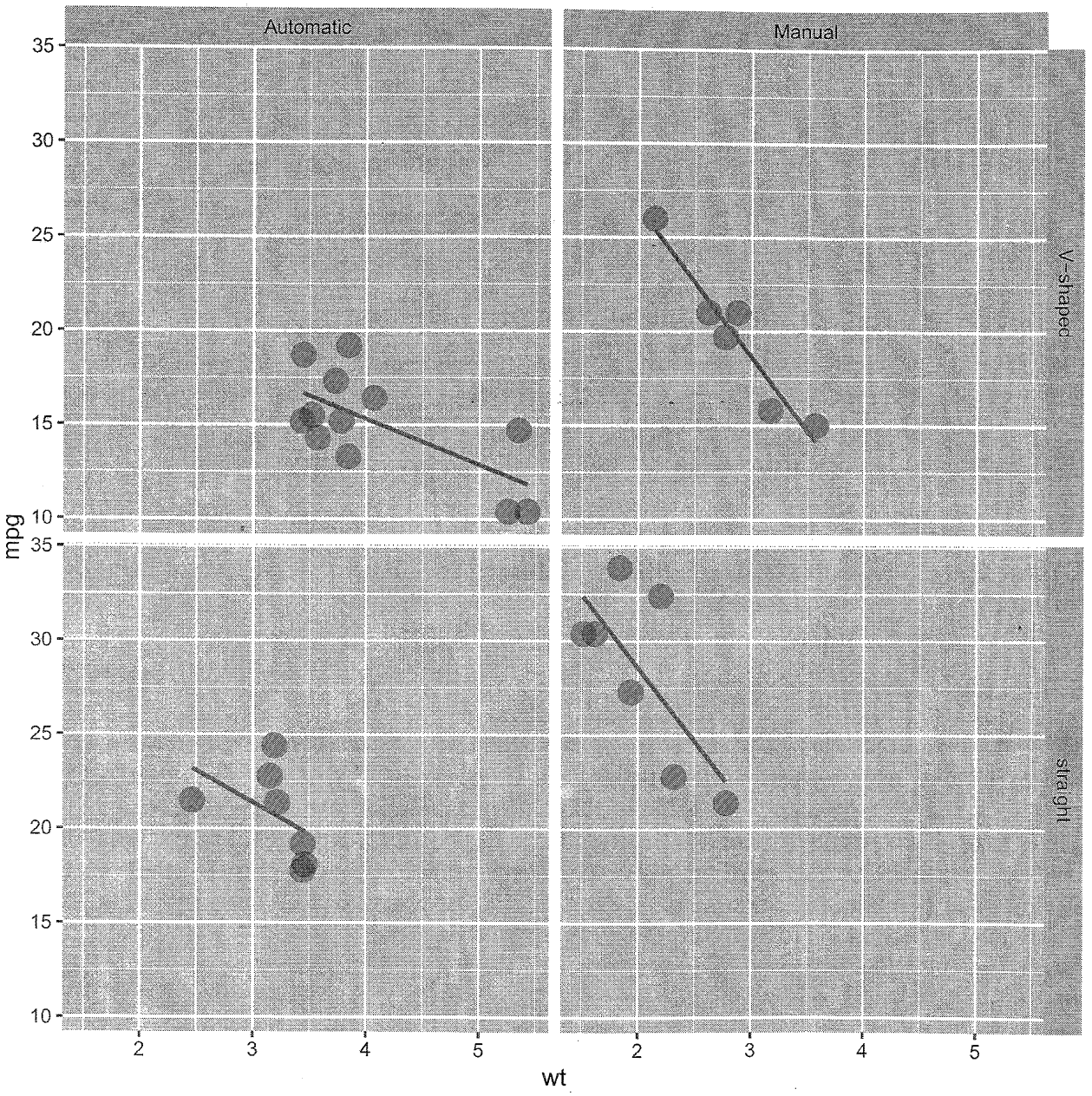

Figure 2.1.8: Faceted scatterplots of miles per gallon against weight by engine and transmission in the `mtcars` dataset.

Although not discussed in the PA modules, it is possible to interweave grouping and faceting, as the following example shows.

Example 2.1.3. (Grouping plus faceting) Write R commands to produce scatterplots of miles per gallon against weight faceted by transmission (`am`). Points in each plot should be color-grouped by engine (`vs`).

Solution. We need two commands to achieve both grouping and faceting.

- To do the coloring according to `vs.fac`, we use the `color` aesthetic in the `ggplot()` call with the command `color = vs.fac`.
- To facet by `am.fac`, we add the command `facet_wrap(~ am.fac)` towards the end of the code.

The resulting commands are given in CHUNK 13 and the faceted scatterplots are shown in Figure 2.1.9. □

```
# CHUNK 13
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, color = vs.fac)) +
  geom_point(size = 5, alpha = 0.4) +
  facet_wrap(~ am.fac)
```

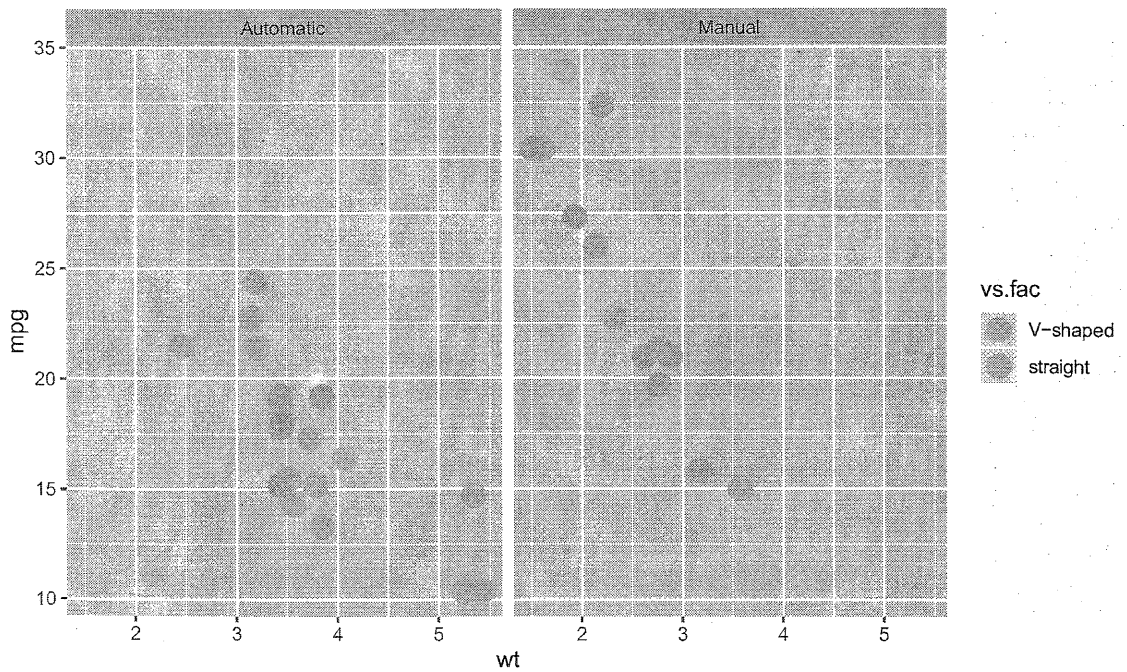

Figure 2.1.9: Faceted scatterplots of miles per gallon against weight by transmission in the `mtcars` dataset. Within each plot, points are color-grouped by engine.

2.1.2 Customizing Your Plots

We learned in the previous subsection how to make a simple but effective ggplot. We now look at how to customize a ggplot in terms of the appearance and range of coordinate axes, and how to add and modify cosmetic enhancements such as legends, titles, and subtitles. You will see that it is easy to fine-tune a ggplot to suit your needs.

Axes. Many datasets have outlier values which, when included in the plots, may distort the scaling of the axes in such a way to obscure the big picture displayed in the data. In other cases, you may want to zoom in and focus on observations lying in a certain range of values, which you can achieve by tweaking the coordinate axes. In `ggplot2`, you can adjust the range of values of the coordinate axes with the `xlim()` and `ylim()` functions, both of which take as an argument a two-element numeric vector indicating the desired lower and upper limits. Data points outside the limits are thrown away and `ggplot2` will give you a warning of how many such points are omitted.

Another way to display the data points more effectively is to adjust the scale, for example, from a linear scale to a log scale. This is especially useful when dealing with highly skewed variables, as we will see in the next section. The function `scale_x_log10()` (do not miss the parentheses!) converts the scale of the x-axis of a ggplot to a log 10 basis and re-positions the data points accordingly. When this function is applied, the points $10, 100, 1000, 10000$ will be shown as consecutive numbers on the x-axis because their log 10 counterparts, $\log_{10} 10 = 1, \log_{10} 100 = 2, \log_{10} 1000 = 3$, and $\log_{10} 1000 = 4$ are consecutive. As you can expect, the function `scale_y_log10()` performs the same operation on the y-axis.

Titles, subtitles, and captions. The `labs()` function allows us to set the label for the x-axis, y-axis, and the text for the title, subtitle, and caption of a ggplot using the `x`, `y`, `title`, `subtitle`, and `caption` arguments, respectively, with the desired label or text supplied as a character string. If you want to set just the label for the x-axis, y-axis, or the text for the title, you can use the `xlab()`, `ylab()`, and `ggtitle()` functions, respectively.

To illustrate the use of these cosmetic enhancements, run CHUNK 14 to produce a version of Figure 2.1.1 with labels for the x-axis, y-axis, and the title added, and with the x-axis restricted to the range between 2 and 4. The resulting scatterplot is shown in Figure 2.1.10. As you can see, 8 observations are excluded from the scatterplot as a result of restricting the range of the x-axis to be between 2 and 4.

Displaying multiple graphs on one page. One thing that differentiates a ggplot from a plot created from the base R installation is that a ggplot can be saved as an object in R and manipulated further. Using the `grid.arrange()` function in the `gridExtra` package (make sure to install the package the first time you use it), we can place several ggplots in a single figure for ease of comparison. Note that this is not the same as faceting, where each figure is for a sub-sample of the data corresponding to the values of the faceting variables. Here, we would like to arrange completely independent ggplots, all of which correspond to the entire sample, side-by-side in a single figure.

As an example, in CHUNK 15 we create two scatterplots, one for miles per gallon against weight and one for miles per gallon against displacement, and place the two plots side-by-side, as in Figure 2.1.11.

```
# CHUNK 14
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point() +
  labs(title = "Automobile Data",
       x = "Weight",
       y = "Miles per Gallon") +
  xlim(2, 4)  # you can also use xlim(c(2, 4))

## Warning: Removed 8 rows containing missing values (geom_point).
```

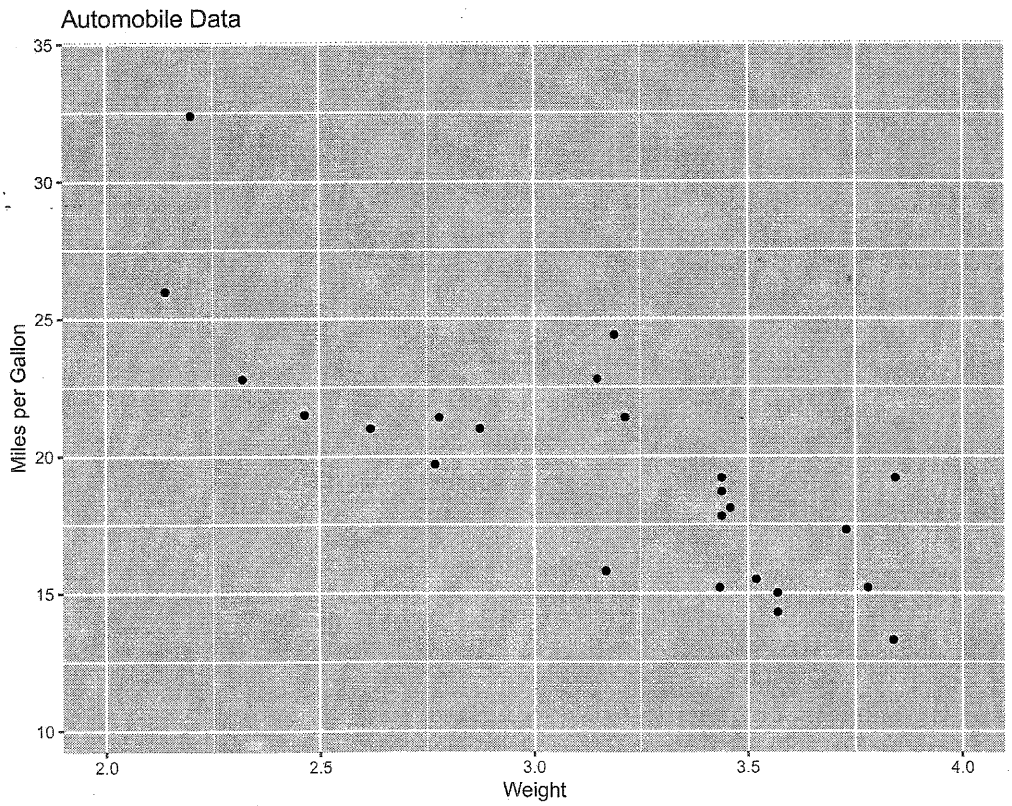

Figure 2.1.10: A version of Figure 2.1.1 with labels for the x-axis, y-axis, and the title added, and with the x-axis restricted to the range between 2 and 4.

```
# CHUNK 15
# uncomment the following line if you haven't installed the package
#install.packages("gridExtra")
library(gridExtra)
p1 <- ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  xlab("Weight") +
  ylab("Miles per Gallon")
p2 <- ggplot(data = mtcars, aes(x = disp, y = mpg)) +
  geom_point() +
  xlab("Displacement") +
  ylab("Miles per Gallon")
grid.arrange(p1, p2, ncol = 2)
```

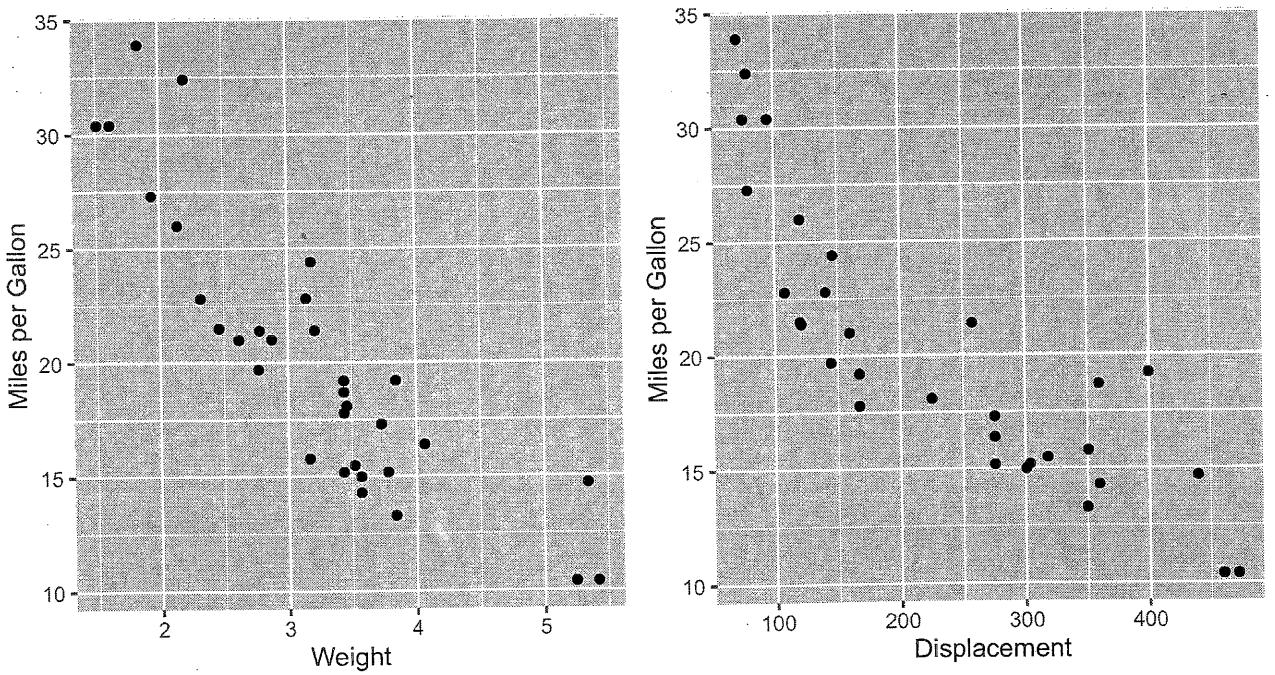

Figure 2.1.11: Two scatterplots, one for miles per gallon against weight and one for miles per gallon against displacement in the `mtcars` dataset, placed side-by-side.